Relational Database Management System 116U01C403

Module 3 .3

Jan 2023-May 2023

Relational Algebra & SQL

Module 3.3

- Set operations, aggregate function, null values Data Manipulation Commands

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 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 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 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' )
```

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 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 QUERIES (cont.)

- In Q12, 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, 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
```

- The original SQL as specified for SYSTEM R also had a **CONTAINS** comparison operator, which is used in conjunction with nested correlated queries
- This operator was dropped from the language, possibly because of the difficulty in implementing it efficiently

CORRELATED NESTED QUERIES (cont.)

- Most implementations of SQL *do not* have this operator
- The CONTAINS operator compares two *sets of values* , and returns TRUE if one set contains all values in the other set
  (reminiscent of the *division* operation of algebra).
  - 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) )
```

CORRELATED NESTED QUERIES (cont.)

- 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

THE EXISTS 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

THE EXISTS FUNCTION (cont.)

- 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)
```

THE EXISTS 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
  - EXISTS is necessary for the expressive power of SQL

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 Relations Feature in SQL2

- 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 Feature in SQL2 (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
```

Joined Relations Feature in SQL2 (cont.)

- could be written as:

```
Q1:   SELECT        FNAME, LNAME, ADDRESS
      FROM  (EMPLOYEE JOIN DEPARTMENT
                ON DNUMBER=DNO)
      WHERE         DNAME='Research'
```

or as:

```
Q1:   SELECT        FNAME, LNAME, ADDRESS
      FROM  (EMPLOYEE NATURAL JOIN DEPARTMENT
                AS DEPT(DNAME, DNO, MSSN, MSDATE)
      WHERE         DNAME='Research'
```

Joined Relations Feature in SQL2 (cont.)

- Another Example;
  - Q2 could be written as follows; this illustrates multiple joins in the joined tables

```
Q2:     SELECT      PNUMBER, DNUM, LNAME,
  BDATE, ADDRESS
        FROM        (PROJECT JOIN
  DEPARTMENT ON DNUM=DNUMBER) JOIN
  EMPLOYEE ON
  MGRSSN=SSN) )
        WHERE       PLOCATION='Stafford'
```

AGGREGATE FUNCTIONS

- Include **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**
- Query 15: Find the maximum salary, the minimum salary, and the average salary among all employees.

```
Q15: 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 16: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

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

AGGREGATE FUNCTIONS (cont.)

- Queries 17 and 18: Retrieve the total number of employees in the company (Q17), and the number of employees in the 'Research' department (Q18).

Q17: SELECT COUNT (*)
FROM EMPLOYEE

Q18: SELECT COUNT (*)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND
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 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
```

  - In Q20, 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 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
```

  - 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 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
```

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

```
Q25:    SELECT    FNAME, LNAME
        FROM          EMPLOYEE
        WHERE     ADDRESS LIKE '%Houston,TX%'
```

SUBSTRING COMPARISON (cont.)

- Query 26: Retrieve all employees who were born during the 1950s. Here, '5' must be the 8th character of the string (according to our format for date), so the BDATE value is '_____5_', with each underscore as a place holder for a single arbitrary character.

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

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

ARITHMETIC OPERATIONS

- The standard arithmetic operators '+', '-'. '*', and '/' (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'
```

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.

```
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 (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

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

Specifying Updates in SQL

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

INSERT

- 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 (cont.)

- Example:

  U1: INSERT INTO EMPLOYEE
  VALUES ('Richard','K','Marini', '653298653', '30-DEC-52',
  '98 Oak Forest,Katy,TX', 'M', 37000,'987654321', 4)

- An alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple
- Attributes with NULL values can be left out
- Example: Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, and SSN attributes.

  U1A: INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)
  VALUES ('Richard', 'Marini', '653298653')

INSERT (cont.)

- Important Note: Only the constraints specified in the DDL commands are automatically enforced by the DBMS when updates are applied to the database
- Another variation of INSERT allows insertion of *multiple tuples* resulting from a query into a relation

INSERT (cont.)

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

```
U3A:    CREATE TABLE  DEPTS_INFO
                (DEPT_NAME     VARCHAR(10),
                 NO_OF_EMPS    INTEGER,
                 TOTAL_SAL     INTEGER);

U3B:    INSERT 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 (cont.)

- Note: The DEPTS_INFO table may not be up-to-date if we change the tuples in either the DEPARTMENT or the EMPLOYEE relations *after* issuing U3B. We have to create a view (see later) to keep such a table up to date.

DELETE

- Removes tuples from a relation
- Includes a WHERE-clause to select the tuples to be deleted
- Tuples are deleted from only *one table* at a time (unless CASCADE is specified on a referential integrity constraint)
- A missing WHERE-clause specifies that *all tuples* in the relation are to be deleted; the table then becomes an empty table
- The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause
- Referential integrity should be enforced

DELETE (cont.)

- Examples:

```
U4A:    DELETE FROM    EMPLOYEE
        WHERE          LNAME='Brown'

U4B:DELETE FROM    EMPLOYEE
        WHERE          SSN='123456789'

U4C:    DELETE FROM    EMPLOYEE
        WHERE          DNO  IN
    (SELECT    DNUMBER
               FROM        DEPARTMENT
               WHERE       DNAME='Research')

U4D:    DELETE FROM    EMPLOYEE
```

UPDATE

- Used to modify attribute values of one or more selected tuples
- A WHERE-clause selects the tuples to be modified
- An additional SET-clause specifies the attributes to be modified and their new values
- Each command modifies tuples *in the same relation*
- Referential integrity should be enforced

UPDATE (cont.)

- Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

```
U5:  UPDATE    PROJECT
     SET       PLOCATION = 'Bellaire', DNUM = 5
     WHERE     PNUMBER=10
```

UPDATE (cont.)

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

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

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