SQL: Advanced Queries, Assertions, Triggers, and Views

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

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 (contd.)

- 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 (contd.)

- In Q12, the nested query has a different result 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
```

CORRELATED NESTED QUERIES (contd.)

- 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
  - 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

CORRELATED NESTED QUERIES (contd.)

- 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 (contd.)

- 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

THE EXISTS FUNCTION (contd.)

- 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 (contd.)

- 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)
```

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 (contd.)

- 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
            EMPLOYEE S ON  E.SUPERSSN=S.SSN)
```

Joined Relations Feature in SQL2 (contd.)

- Examples:

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

- 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 (contd.)

- 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 (contd.)

- 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 (contd.)

- 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 (contd.)

- 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 (contd.)

- 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 (contd.)

- 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
```

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 (contd.)

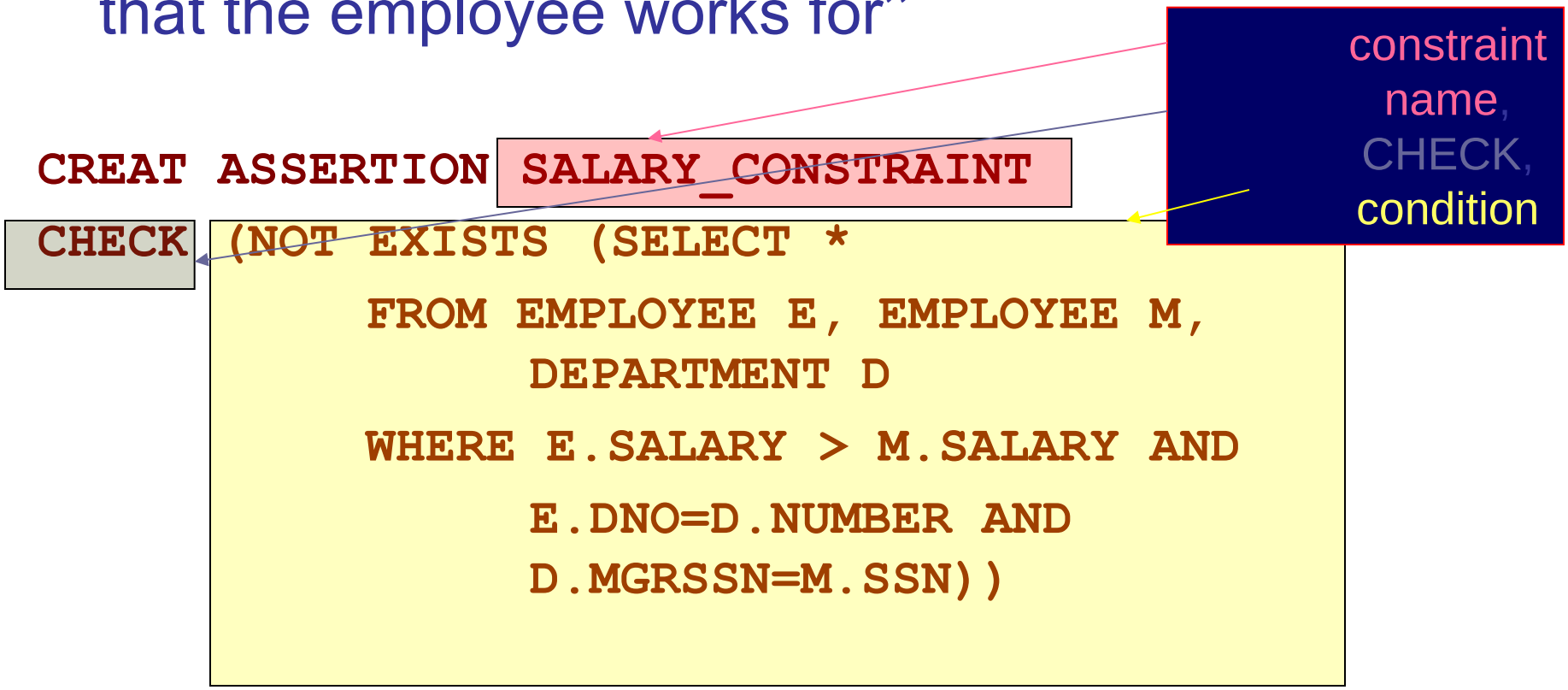
- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- 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

Constraints as 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

Assertions: An Example

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



Using General Assertions

- 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

SQL Triggers

- Objective: to monitor a database and take initiate action when a condition occurs
- Triggers are expressed in a syntax similar to assertions and include the following:
  - Event
    - Such as an insert, deleted, or update operation
  - Condition
  - Action
    - To be taken when the condition is satisfied

SQL Triggers: An Example

- A trigger to compare an employee's salary to his/her supervisor during insert or update operations:

```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF
   SALARY, SUPERVISOR_SSN ON EMPLOYEE
   FOR EACH ROW
       WHEN
       (NEW.SALARY> (SELECT SALARY FROM EMPLOYEE
                     WHERE SSN=NEW.SUPERVISOR_SSN))
       INFORM_SUPERVISOR (NEW.SUPERVISOR_SSN,NEW.SSN);
```

Views in SQL

- A view is a “virtual” table that is derived from other tables
- Allows for limited update operations
  - Since the table may not physically be stored
- Allows full query operations
- A convenience for expressing certain operations

Specification of 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

SQL Views: An Example

- Specify a different WORKS_ON table

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

Using a 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

Efficient View Implementation

- 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

Update Views

- 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