More SQL: Complex Queries, Triggers, Views, and Schema Modification

Part 1

More Complex SQL Retrieval Queries

Comparisons Involving NULL and Three-Valued Logic

- In general, each individual `NULL` value is considered to be different from every other `NULL` value in the various database records.
- When a `NULL` is involved in a comparison operation, the result is considered to be `UNKNOWN` (it may be `TRUE` or it may be `FALSE`).
- Hence, SQL uses a three-valued logic with values TRUE, FALSE, and UNKNOWN instead of the standard two-valued (Boolean) logic with values TRUE or FALSE.
- It is therefore necessary to define the results (or truth values) of three-valued logical expressions when the logical connectives AND, OR, and NOT are used.

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

- In select-project-join queries, the general rule is that only those combinations of tuples that evaluate the logical expression in the `WHERE` clause of the query to `TRUE` are selected.
- Tuple combinations that evaluate to `FALSE` or `UNKNOWN` are not selected.
- However, there are exceptions to that rule for certain operations, such as outer joins.

- SQL allows queries that check whether an attribute value is `NULL`.
- Rather than using = or <> to compare an attribute value to `NULL`, SQL uses the comparison operators `IS` or `IS NOT`.
- This is because SQL considers each `NULL` value as being distinct from every other `NULL` value, so equality comparison is not appropriate.
- It follows that when a join condition is specified, tuples with `NULL` values for the join attributes are not included in the result (unless it is an OUTER JOIN).

- **Query 18**. Retrieve the names of all employees who do not have supervisors.
- Q18:

```
SELECT Fname, Lname
FROM   EMPLOYEE
WHERE  Super_ssn IS NULL;
```

Nested Queries, Tuples, and Set/Multiset Comparisons

- **Query 4**. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

- Q4A:

```
SELECT DISTINCT Pnumber
FROM   PROJECT
WHERE  Pnumber IN (SELECT Pnumber
                   FROM   PROJECT, DEPARTMENT,
                          EMPLOYEE
                   WHERE  Dnum=Dnumber AND
                          Mgr_ssn=Ssn AND
                          Lname='Smith')
       OR
       Pnumber IN (SELECT Pno
                   FROM   WORKS_ON, EMPLOYEE
                   WHERE  Essn=Ssn AND
                          Lname='Smith' );
```

- SQL allows the use of **tuples** of values in comparisons by placing them within parentheses.

- ```
  SELECT DISTINCT Essn
  FROM   WORKS_ON
  WHERE  (Pno, Hours) IN (SELECT Pno, Hours
                          FROM   WORKS_ON
                          WHERE  Essn='123456789');
  ```
  ```

- ```
  SELECT Lname, Fname
  FROM   EMPLOYEE
  WHERE  Salary > ALL (SELECT Salary
                       FROM   EMPLOYEE
                       WHERE  Dno=5);
  ```
  ```

- **Query 16**. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.
- Q16:

```
SELECT E.Fname, E.Lname
FROM   EMPLOYEE AS E
WHERE  E.Ssn IN (SELECT Essn
                 FROM   DEPENDENT AS D
                 WHERE  E.Fname=D.Dependent_name
                        AND
                        E.Sex=D.Sex);
```

- It is generally advisable to create tuple variables (aliases) for *all the tables referenced in an SQL query* to avoid potential errors and ambiguities.

Correlated Nested Queries

- Whenever a condition in the `WHERE` clause of a nested query references some attribute of a relation declared in the outer query, the two queries are said to be **correlated**.
- We can understand a correlated query better by considering that the *nested query is evaluated once for each tuple (or combination of tuples) in the outer query*.

- In general, 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.

- Q16A:

```
SELECT E.Fname, E.Lname
FROM   EMPLOYEE AS E, DEPENDENT AS D
WHERE  E.Ssn=D.Essn AND
       E.Sex=D.Sex AND
       E.Fname=D.Dependent_name;
```

The EXISTS and UNIQUE Functions in SQL

- The `EXISTS` function in SQL is used to check whether the result of a correlated nested query is *empty* (contains no tuples) or not.
- The result of `EXISTS` is a Boolean value **TRUE** if the nested query result contains at least one tuple, or **FALSE** if the nested query result contains no tuples.

- Q16B:

```
SELECT E.Fname, E.Lname
FROM   EMPLOYEE AS E
WHERE  EXISTS (SELECT *
               FROM   DEPENDENT AS D
               WHERE  E.Ssn=D.Essn AND
                      E.Sex=D.Sex AND
                      E.Fname=D.Dependent_name);
```

- **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);
```

- **Query 7**. List the names of managers who have at least one dependent.
- Q7:

```
SELECT Fname, Lname
FROM   EMPLOYEE
WHERE  EXISTS (SELECT *
               FROM   DEPENDENT
               WHERE  Ssn=Essn)
       AND
       EXISTS (SELECT *
               FROM   DEPARTMENT
               WHERE  Ssn=Mgr_ssn );
```

- **Q3**: Retrieve the name of each employee who works on all the projects controlled by department number 5.
- Q3A:

```
SELECT Fname, Lname
FROM   EMPLOYEE
WHERE  NOT EXISTS ((SELECT Pnumber
                    FROM   PROJECT
                    WHERE  Dnum=5)
                    EXCEPT
                   (SELECT Pno
                    FROM   WORKS_ON
                    WHERE  Ssn=Essn));
```

- Q3B:

```
SELECT Lname, Fname
FROM   EMPLOYEE
WHERE  NOT EXISTS (SELECT *
                   FROM   WORKS_ON B
                   WHERE  (B.Pno IN (SELECT Pnumber
                                     FROM   PROJECT
                                     WHERE  Dnum=5)
                           AND
                           NOT EXISTS (SELECT *
                                       FROM   WORKS_ON C
                                       WHERE  C.Essn=Ssn AND
                                              C.Pno=B.Pno)));
```

- There is another SQL function, `UNIQUE(Q)`, which returns `TRUE` if there are no duplicate tuples in the result of query `Q`; otherwise, it returns `FALSE`.
- This can be used to test whether the result of a nested query is a set or a multiset.

Explicit Sets and Renaming of Attributes in SQL

- **Query 17**. Retrieve the Social Security numbers of all employees who work on project numbers 1, 2, or 3.
- Q17:

```
SELECT DISTINCT Essn
FROM   WORKS_ON
WHERE  Pno IN (1, 2, 3);
```

- Q8A:

```
SELECT E.Lname AS Employee_name,
       S.Lname AS Supervisor_name
FROM   EMPLOYEE AS E, EMPLOYEE AS S
WHERE  E.Super_ssn=S.Ssn;
```

Joined Tables in SQL and Outer Joins

- **Q1A:**

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

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

- Q8B:

```
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);
```

- There are a variety of outer join operations.
- In SQL, the options available for specifying joined tables include `INNER JOIN` (only pairs of tuples that match the join condition are retrieved, same as `JOIN`), `LEFT OUTER JOIN` (every tuple in the left table must appear in the result; if it does not have a matching tuple, it is padded with `NULL` values for the attributes of the right table), `RIGHT OUTER JOIN` (every tuple in the right table must appear in the result; if it does not have a matching tuple, it is padded with `NULL` values for the attributes of the left table), and `FULL OUTER JOIN`.
- In the latter three options, the keyword `OUTER` may be omitted.
- If the join attributes have the same name, one can also specify the natural join variation of outer joins by using the keyword `NATURAL` before the operation (for example, `NATURAL LEFT OUTER JOIN`).
- The keyword `CROSS JOIN` is used to specify the CARTESIAN PRODUCT operation, although this should be used only with the utmost care because it generates all possible tuple combinations.

- 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

- **Aggregate functions** are used to summarize information from multiple tuples into a single-tuple summary.
- **Grouping** is used to create subgroups of tuples before summarization.
- A number of built-in aggregate functions exist: **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**.
- These functions can be used in the `SELECT` clause or in a `HAVING` clause.

- **Query 19**. Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.
- Q19:

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

- **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';
```

- **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';
```

- **Query 23**. Count the number of distinct salary values in the database.
- Q23:

```
SELECT COUNT (DISTINCT Salary)
FROM   EMPLOYEE;
```

- **Query 5**: Retrieve the names of all employees who have two or more dependents.
- Q5:

```
SELECT Lname, Fname
FROM   EMPLOYEE
WHERE  (SELECT COUNT (*)
        FROM   DEPENDENT
        WHERE  Ssn=Essn) >= 2;
```

Grouping: The GROUP BY and HAVING Clauses

- **Query 24**. For each department, retrieve the department number, the number of employees in the department, and their average salary.
- Q24:

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

- **Query 25**. For each project, retrieve the project number, the project name, and the number of employees who work on that project.
- Q25:

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

- **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;
```

- **Query 27**. For each project, retrieve the project number, the project name, and the number of employees from department 5 who work on the project.
- Q27:

```
SELECT    Pnumber, Pname, COUNT (*)
FROM      PROJECT, WORKS_ON, EMPLOYEE
WHERE     Pnumber=Pno AND Ssn=Essn AND
          Dno=5
GROUP BY  Pnumber, Pname;
```

- **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
         Dno in (SELECT   Dno
                  FROM     EMPLOYEE
                  GROUP BY Dno
                  HAVING   COUNT (*) > 5)
GROUP BY Dnumber;
```