Lab Session 3 - Advanced SQL Using the PostgreSQL DBMS

By Mitchell Welch

Reading

- Chapter 7 from ***Fundamentals of Database Systems*** by Elmazri and Navathe
- Completing the reading is vital to understand some of the queries!

Summary

- Three Valued Logic
- Nested Select Queries
- The EXISTS and UNIQUE Clauses
- Table Joins
- Exercises For You

House Keeping

- In todays practical session we will be finishing up our tour of PostgreSQL
- This won't be that last we will see of PostgreSQL in the course
    - We will be connecting it up some Java and PHP programs and building databases for them in the upcoming practicals.
- As usual, we will be working with the COMPANY database through our examples, so we will need to create a new database for practical session 3 and load up the data from our week 1 database:
- Create a backup of your prac_01 database using the pg_dump utility:

```
[mwelch8@turing ~]$ pg_dump mwelch8_prac_01 > ~/prac_01.sql
```

- Now create your prac_03 database for this weeks exercise and import your week 1 database using the `\i` command from within the `psql` client:

```
[mwelch8@turing ~]$ createdb mwelch8_prac_03

[mwelch8@turing ~]$ psql mwelch8_prac_03 < ~/prac_01.sql

[mwelch8@turing ~]$ psql mwelch8_prac_03
psql (9.4.4)
Type "help" for help.

...

mwelch8_prac_02=> \dt
              List of relations
 Schema |      Name      | Type  |  Owner
--------+----------------+-------+---------
 public | department     | table | mwelch8
 public | dependent      | table | mwelch8
 public | dept_locations | table | mwelch8
 public | employee       | table | mwelch8
 public | project        | table | mwelch8
 public | works_on       | table | mwelch8
(6 rows)
```

- Throughout this practical session, you should run each of the example queries and review the results returned from the COMPANY database.
- Now you are ready for the prac...

Three Valued Logic

- In SQL it is possible to have NULL-valued attributes.
- The NULL value represents a piece of information that is missing:
  - It may be simply unknown
  - It may be withheld from the database
  - The attribute may not be relevant to the entity that a tuple represents.
    - Because a NULL value represents something that is unknown, NULL values are considered to be different to each other.
    - When NULL values are involved with comparison operations (for example within the attributes in a `WHERE` clause), the result is considered to be `UNKNOWN`.
    - This gives rise to a three-valued system of logic - TRUE, FALSE and UNKNOWN are the values.
    - The results from logical expressions involving AND, OR and NOT operations are summarised below.

Table 5.1 Logical Connectives in Three-Valued Logic

(a)

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

(b)

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

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

- Each section (a,b and c) show the result of applying each operation to a set of two input values.
- For example:
  - (UNKOWN AND TRUE) = UNKNOWN
  - (UNKOWN OR TRUE) = TRUE
  - (NOT UNKNOWN) = UNKNOWN
- In SQL the NULL value can be tested for using the `IS` and `IS NOT` operators. Here is an example for you to try on your COMPANY database:

```
-- Return all employs who do not have supervisors

SELECT fname,lname
FROM EMPLOYEE
WHERE Superssn IS NULL

-- Return all employs who do have supervisors

SELECT fname,lname
FROM EMPLOYEE
WHERE Superssn IS NOT NULL
```

Nested Select Queries

- So far our `SELECT` queries have only involved a single level (i.e. they only contain a single `WHERE` clause)
- Some queries require that existing values in a database be fetched and used in a comparison condition.
- In this situation, we will need to use a *nested query* within the `WHERE` clause of another query.
- The top-level query is referred to as the *outer query*
- Nested queries can be combined with the `IN` and `ALL` operators:

```
-- Returns a list of project numbers that involves
-- an employ with last name 'Smith' as a manager or a worker

SELECT DISTINCT pnumber
FROM project
WHERE pnumber IN
    (SELECT pnumber
    FROM project, department, employee
    WHERE dnum = dnumber AND mgrssn=ssn and lname = 'Smith')
    OR
    pnumber IN
    (SELECT pnumber
    FROM works_on, employee
    WHERE essn=ssn AND lname='Smith');
```

- If the nested query only returns a single tuple with a single attribute, then the result will be a scalar. In this situation you can using the `=` operator for caparison in the outer query
- Sets of attributes can be compared:

```
SELECT DISTINCT essn
FROM works_on
WHERE (pno,hours) IN
    (SELECT pno,hours
    FROM works_on
    WHERE essn = '123456789')
```

- Using the `ALL` operator we can can compare a single value (usually and attribute) to a multi-set of values returned from a nested query:

```
-- Return all employees with a salary greater
-- than all employees in department 5

SELECT fname,lname
FROM EMPLOYEE
WHERE salary > ALL
    (SELECT salary
    FROM employee
    WHERE dno = 5);
```

- When dealing with nested queries, it is possible to have ambiguity within the parameter names between the outer and nested queries.
- The rule of reference is that a reference to an *unqualified* attribute refers to the inner-most nested query.
- To remove ambiguity, we need to qualify the attribute by giving it an alias:

```
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);
```

The EXISTS and Unique Clauses

- The `EXISTS` function in SQL is used to check whether the result of a correlated nested query is empty.
- The result of the `EXISTS` function is the boolean value true if the nested query result contains at least one tuple (FALSE if none are returned).
- The previous query can be re-written:

```
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)
```

- The result of the `EXISTS` can be negated using the `NOT` OPERATOR:

```
-- Retrieve the names of employees who do
-- not have dependents.

SELECT fname, lname
FROM employee
WHERE NOT EXISTS
    (SELECT *
```

```
    FROM dependent
    WHERE ssn=essn);

-- Retrieve the names of employees who do
-- have dependents.

SELECT fname, lname
FROM employee
WHERE EXISTS
    (SELECT *
    FROM dependent
    WHERE ssn=essn);
```

- In the previous examples, if the nested query returns any tuples, the `EXISTS` function will return true. This is negated by the `NOT` clause.
- This query can be extended to name the managers with at least one dependent:

```
SELECT fname, lname
FROM employee
WHERE EXISTS
    (SELECT *
    FROM dependent
    WHERE ssn=essn)
    AND
    EXISTS
        (SELECT *
        FROM department
        WHERE ssn = mgrssn)
```

- In this example we have two nested queries - If there is at lest one in the first and at least one in the second, the employee tuple will be selected.
- The final construct we will look at that uses *universal quantification*. An example of such a query:

```
-- Retrieve the name of each employee who works on all
-- projects by department 4

SELECT lname, fname
FROM employee
WHERE NOT EXISTS
    (SELECT *
    FROM works_on B
    WHERE (B.pno IN
        (SELECT pnumber
        FROM project
        where dnum=4)
        AND
        NOT EXISTS
            (SELECT *
             FROM works_on C
             WHERE C.essn = ssn AND C.pno = B.pno)));
```

- This query can be rephrased as: *Select the employee such that there does not exists a project controlled by department 4 that the employee does not work on*
- What happens if department 4 does not control any projects?

Table Joins

- So far all of the joins in the queries we have looked at have used *equijoins* where a join condition is explicitly defined within the where clause.
- The alternate syntax for achieving a join involves defining a *joined table* in the from clase:

```
SELECT fname,lname,address
FROM (employee JOIN department ON dno=dnumber)
WHERE dname='Research';

-- Original syntax with join through the
-- where clause

SELECT  fname, lname address
FROM employee, department
WHERE dno=dnumber AND dname='Research';
```

- The from clause results in a single joined table.
- Another approach involves using a *Natural Join*
- The *Natural Join* creates an Equijoin between two relations on each pair of attributes that have the same name.

```
SELECT fname, lname, address
FROM (employee NATURAL JOIN department AS DEPT(dname,dno,mssn,msdate))
WHERE dname='Research'
```

- The default type of join used is the *Inner Join*
- In this type of join, tuples are only included in the result if they are linked across both tables.
- This is not always desired - sometimes we might want tuples returned from one table even if they don't have a corresponding record in the other table.
- This is achieved through the use of an *Outer Join*.
- For example, if we wanted to list all employees and their supervises, including those without a supervisor we will use a *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.superssn=s.ssn);
```

- The 'LEFT' type of join indicates that the relation on the left of the expression should have all tuples included
- NULL values are returned for attributes if tuples from the right relation if they are not linked.

Exercises For You

Exercise 1

In SQL construct following queries on the COMPANY database:

a. Retrieve the first name and last name of all employees who do not work on the project with name 'ProductZ'

b. Retrieve the first name and last name of all employees that work on a project that 'John Smith' works on.

c. List the project names of all projects that have least one employee working on them who are supervised by 'John James'.

d. List names of employees that only work on the 'ProductX' project.

Exercise 2

Using the student database that you constructed in exercise 2 of last weeks practical, implement the following SQL queries.

a. Retrieve the names and major departments of all straight-A students (students who have a grade of A in all their courses).

b. Retrieve the names and major departments of all students who do not have any grade of A in any of their courses.

Question 3

Returning to the COMPANY database, use aggregation to construct the following nested queries:

a) Suppose we want the number of male employees in each department rather than all employees (as in Q1 a from prac 2) while retaining a list of departments with average salary per department greater than $30,000 for all employees. Can we specify this query in SQL? Why or why not?

b) A view that has the project name, controlling department name, number of employees, and total hours worked per week on the project for each project with more than one employee working on it.