C9. SQL Subqueries

A subquery is a query places within another query. It is known also as a nested query or an embedded query.

The outer query is named “main” query, when the inner is the subquery.

Subqueries enable writing of queries that select data rows for criteria that are actually developed while the query is executing. This technique support using of dynamic WHERE conditions, computed at run time.

The subquery appears most of the time inside the WHERE clause under following general syntax.

```
SELECT prj_list FROM tables
        WHERE cond_including_subquery;
```

Another possibility is to use a subquery as a temporary table in the FROM clause.

```
SELECT prj_list FROM tables, subqueries
        WHERE join_conditions
                AND where_conditions …;
```

In the where embedded queries, the subquery could be uncorrelated or correlated with the main query. In case of uncorrelated subqueries, the subquery is independent from the main query. It does not include any reference to a filed of the main query and therefore can be resolved independently of the main query.

In case of correlated subqueries, the subquery cannot be resolved independently of the main query. Indeed, it depends of some values (of fields) passed by the main query.

1.1. Subqueries that operate on list of scalar values

A first case is represented by the subqueries that are introduced with the keyword IN. They take the general form for the WHERE clause like:

```
SELECT r.* FROM Relation r
  WHERE expression [NOT] IN (subquery);
```

To understand semantics, think of a nested loops evaluation of main query (outer loop) and a loop for the subquery (inner loop).

ACTIVITY 1: Using SQL Workshop -> SQL Commands run the following SQL command that selects all sailors who have not reserved the boat 103:

```
SELECT  s.sname
        FROM Sailors s
        WHERE  s.sid NOT IN
                (SELECT r.sid
                        FROM Reserves r
                        WHERE r.bid=103);
```

Answer to the following question: how about using just the IN operator in this case?

ACTIVITY 2: In SQL Workshop -> SQL Commands design a query, using the IN operator, that finds all sailors that reserves both Blue and Green boats.

Another set is represented by subqueries that use the keyword ALL. They take the general form a WHERE clause like:

```
SELECT r.* FROM Relation r
  WHERE expression <comp_op> ALL (subquery);
```

The ALL keyword modifies the comparison operator to be applied on all values of the subquery result. If exists at least one value for that comparison fails, the operator will return false.

 **ACTIVITY 3**: Using SQL Workshop -> SQL Commands run the following SQL command that finds all sailors that have the maximum rank:

```
SELECT *
        FROM Sailors
        WHERE rank >= ALL
                (SELECT rank
                        FROM Sailors);
```

Answer to the following question: how about using just the '>' instead of '>=' in this case?

 **ACTIVITY 4**: In SQL Workshop -> SQL Commands design a query, using the ALL operator, which finds the youngest sailors.

Note: be aware of NULL values (x<=NULL return NULL all the time). Solution: remove NULL values from result by using WHERE IS NOT NULL age)

A third example is represented by subqueries that use the keyword ANY. They take the general form for the WHERE clause:

```
SELECT r.*
  FROM Relation r
  WHERE expression <comp_op> ANY (subquery);
```

The ANY keyword is less restrictive than ALL, and it extends the comparison operator only to some values from the set. This means that if a value that return true for comparison operator could be found the ANY operator will return true.

 **ACTIVITY 5**: Using SQL Workshop -> SQL Commands run the following SQL command that finds sailors with rank greater than some sailor called Horace:

```
SELECT *
        FROM Sailors
        WHERE rank > ANY
                (SELECT rank
                        FROM Sailors
                        WHERE name='Horace');
```

Note: if two sailors with the name ‘Horace’ exist, having for example ages 33 and 46, the second one will be included in the result.

1.2. Scalar subqueries

Subqueries that use a comparison operator like equality, inequality, less, less than, greater or greater than. In case that they are used without modifiers, these subqueries must return only a single scalar value. If the subquery returns more than one value the query will fail to execute.

```
SELECT r.*
        FROM Relation r
        WHERE expression <comp_op> (subquery);
```

 **ACTIVITY 6**: Using SQL Workshop -> SQL Commands run the following SQL command that finds all

sailors that are older than the sailor which reserves the boat 103 on {2014-11-23}.

```
SELECT *
        FROM Sailors
        WHERE age >
                (SELECT s.age
                 FROM Sailors s
                        INNER JOIN
                        Reserves r  ON s.sid=r.sid
                 WHERE r.bid=102  AND
                                r.rdate='10/08/2014');
```

1.3. Subqueries that operate on a set of values

These are subqueries that use the EXISTS/NOT EXISTS operators to test the existence/absence of data rows satisfying specified criteria. The subquery in this case can return a scalar or non-scalar set of values.

The general syntax for EXISTS based subqueries is:

```
SELECT r.*
    FROM Relation r
    WHERE [NOT] EXISTS (subquery);
```

The WHERE clause of the outer query tests for the existence of rows returned by the inner query. The subquery does not actually produce any data; rather, it returns a value of TRUE or FALSE.

ACTIVITY 7: Using SQL Workshop -> SQL Commands run the following SQL command that finds all sailors that reserves at least one boat.

```
SELECT *
    FROM Sailors s
    WHERE EXISTS
            (SELECT *
                    FROM Reserves r
                    WHERE r.sid=s.sid);
```

Note: the inner query is correlated with the main query. Indeed, it depends on the actual values of s.sid.

The EXISTS operator could be used to implement also the set division operation. Division identifies attribute values from a relation that are paired with all of the values from another relation.

ACTIVITY 8: Using SQL Workshop -> SQL Commands run the following SQL command that find the names of sailors who have reserved all boats.

```
SELECT s.sid, s.name FROM Sailors s
    WHERE NOT EXISTS
       (SELECT b.bid
            FROM Boats b
            WHERE NOT EXISTS
               (SELECT r.bid
                    FROM Reserves r
                    WHERE r.bid=b.bid AND r.sid=s.sid));
```

ACTIVITY 9: Create an interactive report to display all sailors who have not reserved a boat during a given Year. The report should use a query build using EXISTS or NOT EXISTS operators. For reading the Year add an input field ifYear and a validation button to the report.