Tutorial 4 – SQL Part III
Outer Joins and Correlated Subqueries
(Set Operations, DML)

WITH SUGGESTED SOLUTIONS

Answers to Review Questions

1. Explain the following SQL operations:

   a. *Result table:* every SQL query produces a result in the form of a table – we refer to this table as a result table
   b. *Equi-join:* A join in which the joining condition is based on equality between values in the joining columns.
   c. *Non-equi-join:* A join in which the joining condition is NOT based on equality between values in the joining columns, e.g. >, or `between` condition
   d. *Self-join.* A join that joins a table with itself - often associated with unary relationships, e.g. Manager of Employees.
   e. *Outer join:* The result table includes rows even if the joining condition fails; the column values of the "missing rows" are set to null.
   f. *Union:* SQL union combines the results of two queries – the result table contains rows from both queries eliminating duplicates. Both queries must include the same columns (i.e. drawn from the same domains)

2. Discuss the following:

   a. *Differences between the operation of correlated and uncorrelated subqueries*: In an uncorrelated or Simple Subquery, the innermost subquery executes only once, and the results of this query are stored in a temporary table in memory, and the value(s) in this table is/are used for comparison each time the outer query executes and processes another row with the WHERE condition. So, the subquery in a Simple Subquery can actually be executed as a standalone SELECT statement, it does not depend on or relate to the outer query, i.e. they are not (cor)related.
   However, a Correlated Subquery uses a correlation predicate to reference rows in an outer query from the inner query; the inner subquery executes once for every single row that is processed in the outer query. The execution and the result of the inner query depends on – or is correlated to – the values in the row being processed by the outer query. The inner query's WHERE clause references a value from a table being processed by the outer query. Thus, the inner query cannot be executed independently or as a standalone query. The order of operations in a correlated subquery is as follows: first, a row is processed in the outer query, then, for that particular row (in the outer query), the subquery is executed – so for each row processed by the outer query, the subquery will also be processed.
   b. *Difference between a natural join and an inner join:* if more than one table is listed in the FROM clause of a SELECT statement, and no join condition between the tables is specified, the database engine will try to perform a natural join, i.e. it will check if there are columns in each table that have exactly the same name, and if this is true, it will perform the join between the tables with these columns, checking matching values for each row from the tables.

However, it is always best practice to have a join condition explicitly specified, either as an INNER JOIN, or simply JOIN clause in the FROM clause, or in a WHERE condition, to be sure which columns from the tables will be used in the join operation. This is because if the database engine cannot find common column names in both of the tables, it will perform what is called an ‘extended cross-product’ (see answer below for d.). In the small tables that we’ve used in this subject, you can usually see at a glance if there are common columns, but in most business organisations, there are tables with dozens of columns, and it may be very difficult to ascertain if there are common columns. You don’t really want to discover that there are no common columns between tables by crashing the database server as it tries to execute your query without a specific inner join condition!
The moral of the story is: do not ever have a SELECT statement that queries more than one table without explicitly coding the join condition, relying on natural joins is a big No-No!

c. *Difference between a natural join and a self-join:* a natural join can occur between different tables, whereas a self-join occurs when one is wanting a join condition performed with different rows from the same table. Remember that even though we talk about a Join between tables, and even state the Join condition in terms of tables, the join operation is actually joining Rows of data, not whole tables.

d. *An extended cross-product:* this operation occurs when no join condition is explicitly specified and there are no common columns in the tables and thus a natural join cannot be performed; in this case, the database engine does not know which rows it should be attempting to join, and so joins every single row in table A with every single row in table B, giving M*N rows in the result table. In small tables with for example 10 rows each, this will only result in 100 rows. However, most financial institutions such as banks and insurance companies have tables with hundreds of thousands of rows, so one may end up with a result table of, for example, 10 000*10 000=100 000 000 rows (yes, that’s 100 million!), if the server does not simply keel over and die (crash) before it has finished processing the SELECT – which is how one of my ex-students on their internship discovered to their acute embarrassment that they had forgotten to put in a join condition!

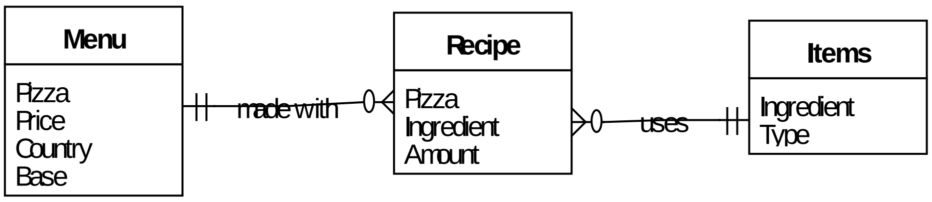
e. *The concept of null values in SQL*: null values represent either missing or not applicable information. Please note that a null value is NOT the same as zero, empty string, nothing; it is an undefined value.

This was not part of the tutorial questions, but tutors stated after the class that some students seemed unsure of when to put a condition into the WHERE clause and when to put a condition into the HAVING clause:
If you are wanting to check the value of a column in an individual row, put the condition into a WHERE clause.
Put a condition into a HAVING clause if you need to check a condition, or value, for a group of rows, such as the number of rows in the group, the average value for the rows in the group etc, i.e. using an aggregate function (see tutorial 4 for examples). If you use a HAVING condition, you must have a GROUP BY clause also.

Solutions to Problems and Exercises

The following SQL exercises continue from tutorial 4 and are based on the Pizza database:

MENU(pizza, price, country, base)
ITEMS(ingredient, type)
RECIPE(*pizza, ingredient*, amount)



OUTER JOINS

1. List **all** fish ingredients and any pizzas that use them. (Give two different ways of constructing this query).

```
select items.ingredient, pizza
from items left outer join recipe
on items.ingredient = recipe.ingredient
where items.type = 'fish'
order by items.ingredient;

select items.ingredient, pizza
from recipe right outer join items
on items.ingredient = recipe.ingredient
where items.type = 'fish'
order by items.ingredient;
```

Result

```
ingredient | pizza
----------- +-----------
anchovies | napolitana
anchovies | siciliano
anchovies | special
anchovies | stagiony
prawn |
seafood | seafood
seafood | special
(7 rows)
```

2. List all fish ingredients and all pizzas – those pizzas that include fish ingredients and those that do not.

   The below query produce all fish ingredient and any pizza that use fish ingredient.

```
select items.ingredient, pizza
from recipe full outer join items
on items.ingredient = recipe.ingredient
where items.type = 'fish';
```

Result

```
ingredient | pizza
------------ +------------
anchovies | napolitana
anchovies | siciliano
anchovies | special
anchovies | stagiony
prawn |
seafood | seafood
seafood | special
(7 rows)
```

Please note that this question may be interpreted as to list all pizza and all fish ingredient. The following is the query for this interpretation.

```
select distinct  pizza, i.ingredient from
recipe r full outer join (select ingredient, type from items where type='fish')i
on r.ingredient= i.ingredient;
```

```
pizza    | ingredient
------------+------------
 americano  |
 cabanossi  |
 garlic     |
 ham        |
 hawaiian   |
 margarita  |
 mexicano   |
 mushroom   |
 napolitana | anchovies
 napolitana |
 seafood    | seafood
 seafood    |
 siciliano  | anchovies
 siciliano  |
 special    | anchovies
 special    | seafood
 special    |
 stagiony   | anchovies
 stagiony   |
 vegetarian |
            | prawn
(21 rows)
```

CORRELATED SUBQUERY OPERATIONS

3. List each ingredient and the pizza that contains the largest amount of this ingredient (use correlated subquery operation).

```
select ingredient, pizza, amount
from recipe r
where amount = (select max(amount)
                from recipe
                where ingredient = r.ingredient);
```

Result

```
ingredient | pizza | amount
------------ +------------ +--------
cheese | margarita | 120
ham | ham | 150
anchovies | napolitana | 100
olives | napolitana | 75
pineapple | hawaiian | 100
cabanossi | cabanossi | 150
capsicum | siciliano | 75
salami | americano | 120
pepperoni | americano | 75
onion | mexicano | 75
capsicum | mexicano | 75
chilli | mexicano | 25
spice | mexicano | 20
seafood | seafood | 200
garlic | garlic | 25
peas | vegetarian | 50
tomato | vegetarian | 50
mushroom | mushroom | 100
bacon | special | 25
egg | special | 25
 (20 rows)
```

4. List ingredients used in more than one pizza (use correlated subquery operation).

```
select distinct ingredient
from recipe r
where ingredient in (select ingredient
                     from recipe
                     where pizza <> r.pizza);
```

Result

```
ingredient
------------
anchovies
cabanossi
capsicum
```

```
cheese
ham
mushroom
olives
onion
peas
pineapple
salami
seafood
spice
tomato
(14 rows)
```

5. Which ingredients are not used in any pizza? (use correlated subquery operation)

```
select ingredient
from items i
where not exists (select *
                  from recipe r
                  where i.ingredient = r.ingredient);
```

Result

```
ingredient
------------
prawn
(1 row)
```

6. Which ingredients are not used in any pizza? (use uncorrelated/simple subquery operation)

```
select ingredient
from items
where ingredient not in (select ingredient from recipe);
```

Result

```
ingredient
------------
prawn
(1 row)
```

INSERT, UPDATE AND DELETE STATEMENTS

1. Insert a row into the Pizza table with the following values: ('bondi',6.20,'australia','wf')

```
insert into menu (pizza,price,country,base) values ('bondi',6.20,'australia','wf')
```

2. Increase the PRICE of the Bondi pizza by 10%.

```
update menu
set price = price*1.1
where pizza = 'bondi';
```

3. Delete the Bondi pizza.

```
delete
from menu
where pizza = 'bondi';
```