Table of Contents

As you develop more complex queries you will want to consider setting up a test plan to help decide if your query is working correctly. The first- and sometimes the hardest - step is to figure out what the task asks for. I have been trying to write the tasks using more of an English approach than an SQL approach. For example, I ask you to display the client's last name, not the cl_name_last attribute. Often setting up a test plan will help you understand the task description. The next step could be to set up a few sets of data that meet (or do not meet) the task description and state what result you expect. Now you have to decide what to do with that data. You might think of adding the data sets to tables we already have- which can create its own problems. Or you can set up a few tables that are very similar to the database tables we already have, but simplified.

Let's take a few of the query patterns from the previous documents and work out some tasks related to these patterns. These are some basic patterns using the vets tables.

1. We want clients at the vet clinic who have a hedgehog.
2. We want clients at the vet clinic who have both a hedgehog and a porcupine.
3. We want clients at the vet clinic who have a hedgehog but not a porcupine.
4. We want clients at the vet clinic who have a hedgehog and a porcupine and a dormouse.
5. We want clients at the vet clinic who have (a hedgehog or a porcupine) and a dormouse.
6. We want clients at the vet clinic who have (a hedgehog or a porcupine) but not a dormouse.
7. We want clients at the vet clinic who do not have a hedgehog.

1. Getting started with test data

What we want to do here is start with some data that we could use to check our understanding of the question to be answered. We need to include data that does not exist in our vets tables. Maybe we don't have any clients with a hedgehog!

I started with a simple question- we want clients who have a hedgehog. Assume we will have to display the client id and name. We will know if a client has a hedgehog if the client's id is in the animal table for a row where an_type = 'hedgehog'.

The an_type is not null in the animals table. We do not have to worry about a possibility of a client having an animal but we do not know what type of animal it is.

So for each client there are two possibilities- either they have a hedgehog or they do not have a hedgehog. We do not care how many hedgehogs they might have- just if they have at least one or if they do not have any hedgehogs. (It might be a good idea to check this assumption.)

We can start with a simple tabular display- These are easy to set up in a spreadsheet program- or on a piece of paper. These are called truth tables and are very helpful in thinking about a question.

We have only one logical predicate: 'This client has a hedgehog".

P1 = this client has a hedgehog.

This is our table of possibilities. The simplest tables generally look like they are too easy to bother with. But start easy.

P1
TRUE
FALSE

Question 2 wants clients with both a hedgehog and a porcupine. That involves two tests. I will call these predicates P2A and P2B
P2A = this client has a hedgehog.
P2B = this client has a porcupine.
P2 = this client has both a hedgehog and a porcupine
P2 = P2A AND P2B

Since there are 2 possible values for P2A and 2 possible values for P2B, there are 4 possible combinations of values.

P2A	P2B	P2A AND P2B
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

Now you are probably thinking about sql- stop that and think about how many clients to we need to have for testing.

For question 1 we need two clients- one with a hedgehog and one without a hedgehog.

For question 2 we need four clients- one with a hedgehog and a porcupine, one with a hedgehog and no porcupine, one with a porcupine and no hedgehog, and one with neither a hedgehog, not a porcupine.

We can use the same set of record for question 3.
P3A = this client has a hedgehog.
P3B = this client has a porcupine.
P3 = this client has a hedgehog and this client does not have a porcupine
P3 = P3A AND (NOT P3B)

P3A	P3B	NOT P3B	P3A AND (NOT P3B)
TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE

Now you need to imagine that someone is going to write the query and they do not write very good SQL. Perhaps they write the query to test for exactly one hedgehog. So we might want a few more rows of tests-
a client with 4 hedgehogs and 2 porcupines
a client with 3 hedgehogs and no porcupines
a client with only one dog.
a client with no animals at all

When I get to the questions that test for hedgehogs, porcupines and dormice, then I will have a bigger table.
P4A = this client has a hedgehog.
P4B = this client has a porcupine.
P4C = this client has a dormouse.
P4 = this client has a hedgehog and a porcupine and a dormouse
P4 = P4A AND P4B AND P4C
P5 = ?
P6 = ?

This is 8 possible patterns

	P4A	P4B	P4C	P4	P5	P6
1	TRUE	TRUE	TRUE	TRUE		
2	TRUE	TRUE	FALSE	FALSE		
3	TRUE	FALSE	TRUE	FALSE		
4	TRUE	FALSE	FALSE	FALSE		
5	FALSE	TRUE	TRUE	FALSE		
6	FALSE	TRUE	FALSE	FALSE		
7	FALSE	FALSE	TRUE	FALSE		
8	FALSE	FALSE	FALSE	FALSE		

2. Test Tables and Rows

Now I could start adding rows to my tables for vt_clients and vt_animals- but those tables have a lot of columns I really do not care about- there is nothing in these questions that needs to know what city the client lives in or the animal dob. So I am going to set up two test tables. The test table for the animals will have attributes for an_id, cl_id, and an_type. The test table for the clients will have a cl_id attribute and an attribute for client name. I need to use data types to match those in the vt_clients and vt_animals tables. I also need to keep the relationship between these two tables.

I want to keep the tables as simple as possible- but no simpler.

Demo 01: I kept the pk and fk constraints and any constraint on those attributes.

```
create table tst_clients(
    cl_id           int
  , cl_name_last    varchar(25)     not null
  , constraint tst_clients_pk primary key (cl_id)
);

create table tst_animals(
    an_id           int
  , an_type         varchar(25)     not null
  , cl_id           int             not null
  , constraint tst_animals_clients_fk foreign key (cl_id) references tst_clients(cl_id)
  , constraint tst_animals_pk primary key (an_id)
  );
```

Demo 02: Inserts - I am using the client name as a code to the animals for that client. The name 'H__P__D__' means this is a client with a Hedgehog and a porcupine and a Dormouse. If you look at some of the sample runs, you can see that the underscores make that column easier to read and scan.

```
insert into tst_clients values (1, 'H__P__D__');
  insert into tst_animals values (10, 'hedgehog', 1);
  insert into tst_animals values (11, 'porcupine', 1);
  insert into tst_animals values (12, 'dormouse', 1);

insert into tst_clients values (2, 'H__P_____');
  insert into tst_animals values (13, 'hedgehog', 2);
  insert into tst_animals values (14, 'porcupine', 2);
```

```
insert into tst_clients values (3, 'H_____D__');
  insert into tst_animals values (15, 'hedgehog', 3);
  insert into tst_animals values (16, 'dormouse', 3);

insert into tst_clients values (4, 'H________');
  insert into tst_animals values (17, 'hedgehog', 4);

insert into tst_clients values (5, '___P__D__');
  insert into tst_animals values (18, 'porcupine', 5);
  insert into tst_animals values (19, 'dormouse', 5);

insert into tst_clients values (6, '___P_____');
  insert into tst_animals values (20, 'porcupine', 6);

insert into tst_clients values (7, '______D__');
  insert into tst_animals values (21, 'dormouse', 7);

insert into tst_clients values (8, '_________');

-- And a few additional inserts
insert into tst_clients values (9, 'H4_P2____');
  insert into tst_animals values (22, 'hedgehog', 9);
  insert into tst_animals values (23, 'hedgehog', 9);
  insert into tst_animals values (24, 'hedgehog', 9);
  insert into tst_animals values (25, 'hedgehog', 9);
  insert into tst_animals values (26, 'porcupine', 9);
  insert into tst_animals values (27, 'porcupine', 9);

insert into tst_clients values (10, 'H3_______');
  insert into tst_animals values (28, 'hedgehog', 10);
  insert into tst_animals values (29, 'hedgehog', 10);
  insert into tst_animals values (30, 'hedgehog', 10);

insert into tst_clients values (11, 'dog');
  insert into tst_animals values (31, 'dog', 11);
```

Before you start writing and running queries, decide what you want the result to look like. It is very easy to just be happy to get any result- you need the correct result.

```
select * from tst_clients;
```

```
      cl_id cl_name_last
----------- -------------------------
          1 H__P__D__
          2 H__P_____
          3 H_____D__
          4 H________
          5 ___P__D__
          6 ___P_____
          7 ______D__
          8 _________
          9 H4_P2____
         10 H3_______
         11 dog
```

Note that client 8 does not come back with our joined tables because that client has no animals.

```
select cl.cl_id, cl_name_last, an_id, an_type
from tst_clients cl
join tst_animals an  on cl.cl_id = an.cl_id
;
```

```
      cl_id cl_name_last                    an_id an_type
----------- ------------------------ ----------- -------------------------
          1 H__P__D__                         10 hedgehog
          1 H__P__D__                         11 porcupine
          1 H__P__D__                         12 dormouse
          2 H__P_____                         13 hedgehog
          2 H__P_____                         14 porcupine
          3 H_____D__                         15 hedgehog
          3 H_____D__                         16 dormouse
          4 H________                         17 hedgehog
          5 ___P__D__                         18 porcupine
          5 ___P__D__                         19 dormouse
          6 ___P_____                         20 porcupine
          7 ______D__                         21 dormouse
          9 H4_P2____                         22 hedgehog
          9 H4_P2____                         23 hedgehog
          9 H4_P2____                         24 hedgehog
          9 H4_P2____                         25 hedgehog
          9 H4_P2____                         26 porcupine
          9 H4_P2____                         27 porcupine
         10 H3_______                         28 hedgehog
         10 H3_______                         29 hedgehog
         10 H3_______                         30 hedgehog
         11 dog                               31 dog

22 rows selected
```

3. Queries

Demo 03: Question 1: we should get cl_ids 1,2,3,4,9,10

```
select cl.cl_id, cl.cl_name_last
from tst_clients cl
join tst_animals an  on cl.cl_id = an.cl_id
where an_type = 'hedgehog'
;
```

```
      cl_id cl_name_last
----------- ------------------------
          1 H__P__D__
          2 H__P_____
          3 H_____D__
          4 H________
          9 H4_P2____
          9 H4_P2____
          9 H4_P2____
          9 H4_P2____
         10 H3_______
         10 H3_______
         10 H3_______
```

There is no reason to display a client multiple times (such as client 9). We could fix this with Distinct in the select or by using a subquery. Since we are talking about subqueries, we will use that approach. The columns I want to display are both in the clients table so that is the table in the main query.

Demo 04: Question 1: we should get cl_ids 1,2,3,4,9,10. Note that the subquery is P1 with a value of True. Assuming I did my cl_name_last values correctly, I can scan down these rows and see 'H' is each column. I should also check that the rows that are not displayed are client which do not have hedgehogs. In this case I might just with to compare the list for the entire table and check the clients no returned. It is easy to forget the "negative test".

```
select cl.cl_id, cl.cl_name_last
from tst_clients cl
where cl.cl_id in (
    select cl_id
    from tst_animals
    where an_type = 'hedgehog');
```

```
     cl_id cl_name_last
---------- ------------------------
         1 H__P__D__
         2 H__P_____
         3 H_____D__
         4 H________
         9 H4_P2____
        10 H3_______
```

Demo 05: Question 2: We want clients who have both a hedgehog and a porcupine; we should get cl_ids 1,2,9. The client needs to pass two tests: P2A and P2B. Visually compare fpor the negative test- client 3 does not show up- it that correct?

```
select cl.cl_id, cl.cl_name_last
from tst_clients cl
where cl.cl_id in (
    select cl_id
    from tst_animals
    where an_type = 'hedgehog')
and cl_id in (
    select cl_id
    from tst_animals
    where an_type = 'porcupine');
```

```
     cl_id cl_name_last
---------- ------------------------
         1 H__P__D__
         2 H__P_____
         9 H4_P2____
```

At this point you might wish to consider the tasks and determine the rows that should be returned **before** you write the query. It is very easy to assume the query is correct if it runs.

Demo 06: Question 3: We want clients who have a hedgehog and not a porcupine; we should get cl_ids 3,4,10. The client needs to pass two tests: P2A and P2B.

```
select cl.cl_id, cl.cl_name_last
from tst_clients cl
where cl.cl_id in (
    select cl_id
    from tst_animals
    where an_type = 'hedgehog')
and cl_id not in (
    select cl_id
    from tst_animals
    where an_type = 'porcupine')
;
```

```
     cl_id cl_name_last
---------- ------------------------
         3 H_____D__
         4 H________
        10 H3_______
```

Demo 07: Question 4 is very much like question 2. Three predicates; three subqueries.

```
select cl_id, cl_name_last
from tst_clients
where
? ? ?
```

Demo 08: Question 5: We want clients who have (a hedgehog or a porcupine) and a dormouse. we should get cl_ids 1,3,5 Three predicates; three subqueries; take care with the order of precedence.

```
select cl_id, cl_name_last
from tst_clients
where
(
    cl_id in (
        select cl_id
        from tst_animals
        where an_type = 'hedgehog')
    OR cl_id in (
        select cl_id
        from tst_animals
        where an_type = 'porcupine')
)
and cl_id in (
    select cl_id
    from tst_animals
    where an_type = 'dormouse')
;
```

```
     cl_id cl_name_last
---------- -------------------------
         1 H__P__D__
         3 H_____D__
         5 ___P__D__
```

Demo 09: Question 6: We want clients who have (a hedgehog or a porcupine) and NOT a dormouse. We should get cl_ids 2, 4, 6, 9, 10 Three predicates; three subqueries; take care with the order of precedence.

```
select  cl_id, cl_name_last
from tst_clients
where
(
    cl_id in (
        select cl_id
        from tst_animals
        where an_type = 'hedgehog')
    OR cl_id in (
        select cl_id
        from tst_animals
        where an_type = 'porcupine')
)
AND cl_id NOT in (
    select cl_id
    from tst_animals
    where an_type = 'dormouse')
;
```

```
     cl_id cl_name_last
---------- -------------------------
         2 H__P_____
         4 H________
         6 ___P_____
         9 H4_P2____
        10 H3_______
```

Demo 10: Question 7 We want clients who do not have a hedgehog. We should get cl_id.5, 6, 7, 8, 11

```
select cl_id, cl_name_last
from tst_clients
where
    cl_id NOT in (
        select cl_id
        from tst_animals
        where an_type = 'hedgehog');
```

```
     cl_id cl_name_last
---------- -------------------------
         5 ___P__D__
         6 ___P_____
         7 ______D__
         8 _________
        11 dog
```

Since I kept the attribute names and types matching those of the vt_clients and vt_animals tables, I should be able to simple adjust the table name to run against those tables.

4. Things you should NOT do for these patterns

It is always dangerous to show you queries that do not do the processing we want. Sometimes people just copy these as demos. These are **incorrect** logic for question 2.

The following gets clients with either a hedgehog or a porcupine. This is not a multi-match pattern. Using an In filter {`where an_type IN ('hedgehog', 'porcupine')`} doesn't help since an IN list is an OR test.

```
select distinct cl.cl_id, cl.cl_name_last
from tst_clients cl
join tst_animals an  on cl.cl_id = an.cl_id
where an_type = 'hedgehog'
or    an_type = 'porcupine'
order by cl_id;
```

```
     cl_id cl_name_last
---------- -------------------------
         1 H__P__D__
         2 H__P_____
         3 H_____D__
         4 H________
         5 ___P__D__
         6 ___P_____
         9 H4_P2____
        10 H3_______
```

The following gets no rows since you are testing for an animal that is both a porcupine and a hedgehog at the same time. Each of our animals has only one value for an_type. The way this is written, it uses a row filter testing one row at a time.

```
select distinct cl.cl_id, cl.cl_name_last
from tst_clients cl
join tst_animals an  on cl.cl_id = an.cl_id
where an_type = 'hedgehog'
and   an_type = 'porcupine'
order by cl_id
;
```

```
no rows selected
```

The following gets no rows because it is also testing for an animal that has two types at the same time. Note that here the subquery test is on the an_id; it should be on the cl_id. We want clients who have a hedgehog and a porcupine, not animals that are a hedgehog and a porcupine.

```
select distinct cl.cl_id, cl.cl_name_last
from tst_clients cl
join tst_animals an  on cl.cl_id = an.cl_id
where an_id in (
    select  an_id
    from tst_animals
    where an_type = 'hedgehog')
and an_id in (
    select  an_id
    from tst_animals
    where an_type = 'porcupine')
;
```

```
no rows selected
```