Table of Contents

There is another type of query- a multi-match query- that we will see implemented in various ways over the semester. We have looked at this pattern already in a simpler format.

This is not an easy topic. You should read through this once- sort of as a story. Then go back and study this. This keeps popping up this semester often. You might want to read the last section of this document first.

1. Departments and employees

The Where clause we have been using is called a row filter because it is used to examine the data in the From clause's virtual table **one row at a time** to see if that row will be allowed in the final result.

We are going to start with the employee table. This table is set up with the assumption that each employee is identified by an employee id; that is the primary key for the employee table. That means each employee gets exactly one row in the employee table. The definition of a row in the employee table has one attribute for the employee's job (job_id) which is a not null column and one attribute for dept_id which is also not null. This means that for each employee we store only one value for job_id, the employee's current job and one value for dept_id, the employee's current department.

Demo 01: We have currently 22 employees and each employee has a job id value and a dept_id.

```
+--------+---------+--------+
| emp_id | dept_id | job_id |
+--------+---------+--------+
|    100 |      10 |      1 |
|    201 |      20 |      2 |
|    101 |      30 |     16 |
|    108 |      30 |     16 |
|    109 |      30 |     32 |
|    110 |      30 |     32 |
|    203 |      30 |     16 |
|    204 |      30 |     32 |
|    205 |      30 |     16 |
|    206 |      30 |     32 |
|    162 |      35 |     16 |
|    200 |      35 |     16 |
|    207 |      35 |      8 |
|    145 |      80 |      4 |
|    150 |      80 |      8 |
|    155 |      80 |      8 |
|    103 |     210 |     64 |
|    104 |     210 |     32 |
|    102 |     215 |     64 |
|    146 |     215 |     64 |
|    160 |     215 |     32 |
|    161 |     215 |     16 |
+--------+---------+--------+
```

These are the rows in the departments table. Note that dept_id 90 and 95 do not have any employees and were not returned by the joined query..

```
+---------+-----------------+
| dept_id | dept_name       |
+---------+-----------------+
|      10 | Administration  |
|      20 | Marketing       |
|      30 | Development     |
|      35 | Cloud Computing |
|      80 | Sales           |
|      90 | Shipping        |
|      95 | Logistics       |
|     210 | IT Support      |
|     215 | IT Support      |
+---------+-----------------+
```

Demo 02: We can filter for employees with job_id 16 and each row in the employees table will be looked at to see if the job_id for that row is 16 or not.

```
select emp_id, dept_id, job_id
from employee.employees
where job_id = 16
order by dept_id, emp_id;
+--------+---------+--------+
| emp_id | dept_id | job_id |
+--------+---------+--------+
|    101 |      30 |     16 |
|    108 |      30 |     16 |
|    203 |      30 |     16 |
|    205 |      30 |     16 |
|    162 |      35 |     16 |
|    200 |      35 |     16 |
|    161 |     215 |     16 |
+--------+---------+--------+
7 rows in set (0.00 sec)
```

Demo 03: We can use the Or operator to find employees with job id 16 or with job id 32. We could also use an IN list

```
select emp_id, dept_id, job_id
from employee.employees
where job_id = 16 or job_id = 32
order by emp_id;
+--------+---------+--------+
| emp_id | dept_id | job_id |
+--------+---------+--------+
|    101 |      30 |     16 |
|    104 |     210 |     32 |
|    108 |      30 |     16 |
|    109 |      30 |     32 |
|    110 |      30 |     32 |
|    160 |     215 |     32 |
|    161 |     215 |     16 |
|    162 |      35 |     16 |
|    200 |      35 |     16 |
|    203 |      30 |     16 |
|    204 |      30 |     32 |
|    205 |      30 |     16 |
|    206 |      30 |     32 |
+--------+---------+--------+
13 rows in set (0.00 sec)
```

Demo 04: If we use the AND operator to find employees with job id 16 and with job id 32 we get no rows returned because each row in the employee table is looked at one row at a time, and no row has the value 16 and the value 32 for job id at the same time.

```
select emp_id, dept_id, job_id
from employee.employees
where job_id = 16 AND job_id = 32
order by emp_id;
```

```
Empty Set
```

1.1. Multi-matches

Now we want to look at employees and their jobs from a department point of view.

Demo 05: Suppose we want to see which departments have an employee with job id 16. We can do this with the filter we had earlier, show just the department id and use distinct to remove duplicate rows- these would be departments where we have more than one employee with job 16.

```
select distinct dept_id
from employee.employees
where job_id = 16 ;
+---------+
| dept_id |
+---------+
|      30 |
|      35 |
|     215 |
+---------+
```

Demo 06: Same query for departments with employees with job 32

```
select distinct dept_id
from employee.employees
where job_id = 32 ;
+---------+
| dept_id |
+---------+
|      30 |
|     210 |
|     215 |
+---------+
```

Demo 07: This gives us departments where we have employees who have either job 16 or job 32. Compare this result with the results of the previous two queries.

```
select distinct dept_id
from employee.employees
where job_id = 16  or job_id = 32;
+---------+
| dept_id |
+---------+
|      30 |
|      35 |
|     210 |
|     215 |
+---------+
```

But suppose we try to use the AND operator to find departments which have both employees with job id 16 and employees with job id 32. We get no rows for the same reason we did in the earlier AND filter since no employee has two different job id values. So the And test is never met.

Demo 08:

```
select distinct dept_id
from employee.employees
where job_id = 16  AND job_id = 32
;
```

```
Empty Set
```

This is one of those rules that seems trivial when you read it, but it causes all kinds of errors on assignments and on exams. If you are looking at a single table as the table expression, each cell has only a value. The job_id cell cannot have two different values **in the same row**.

We have to approach this from another way. We are still thinking about row filters, but we are asking the question about the **departments** so we start with the department table. Our query starts with a row from the department table and asks two questions
(1) are you a department that has an employee with job id 16
and (2) are you a department that has an employee with job id 32.

These are two separate questions. If this department row meets both tests then it is passed into the result. We are asking two questions here and both questions have to be met - that is why I am calling these multiple-match queries.

Demo 09: Compare this result to the previous ones. This query says show me the departments where the department has at least one employee with job 16 and the department has at least one employee with job 32. The department has to pass both of these tests to get into the output.
This uses both the department table and the employee table.

```
select dept_id
from employee.departments
where dept_id in (
   select dept_id
   from employee.employees
   where job_id = 16 )
and dept_id in (
   select dept_id
   from employee.employees
   where job_id = 32 );
+---------+
| dept_id |
+---------+
|      30 |
|     215 |
+---------+
```

There is a version in the demos that uses only the employee table, but I think this version will be clearer for some other examples.

You should be able to think of how to write a query for the following:

Show the departments that have employees with job id 16 and employees with job id 32 and employees with job id 8.

Show the departments that have employees with job id 16 and employees with either job id 32or job id 8.

1.2. Non-matches and mixed matches

Now let's try queries that do non-matches- find department which do not have an employee with job_id 16. Before we start we need to think about department 90 and 95; they do not have any employees. Should they be returned by our query? In one sense- they should- if the department has no employees then it certainly has no employees with job_id 16. But that might not be the intent- maybe we really are supposed to find department with employees but not with employees with job_id 16. You probably should verify what is actually wanted with the person asking you to write the query.

Demo 10: Compare the following two queries and figure out why one returns dept 90 and 95 and the other doesn't and figure out why I included Distinct in one of these and not in the other. If you have troubles with this- ask in the forum. Check the use of tables in the various clauses.

```
select dept_id
from employee.departments
where dept_id NOT IN (
   select dept_id
   from employee.employees
   where job_id = 16 )
order by dept_id;
+---------+
| dept_id |
+---------+
|      10 |
|      20 |
|      80 |
|      90 |
|      95 |
|     210 |
+---------+
6 rows in set
```

Demo 11:

```
select distinct dept_id
from employee.employees
where dept_id NOT IN (
   select dept_id
   from employee.employees
   where job_id = 16 )
order by dept_id;
+---------+
| dept_id |
+---------+
|      10 |
|      20 |
|      80 |
|     210 |
+---------+
4 rows in set
```

Demo 12: Why is this query incorrect for our task? What is this query actually returning? If you have troubles, check what the inner subquery returns.

```
select distinct dept_id
from employee.employees
where dept_id  IN (
   select dept_id
   from employee.employees
   where job_id !=16 )
order by dept_id
;
+---------+
| dept_id |
+---------+
|      10 |
|      20 |
|      30 |
|      35 |
|      80 |
|     210 |
|     215 |
+---------+
```

Demo 13: Show the departments that have employees with job id 16 and have no employees with job id 32.

```
select dept_id
from employee.departments
where dept_id  IN (
   select dept_id
   from employee.employees
   where job_id = 16 )
and  dept_id  NOT IN (
   select dept_id
   from employee.employees
   where job_id = 32 )
order by dept_id;
+---------+
| dept_id |
+---------+
|      35 |
+---------+
```

These are not different subqueries than used in the previous documents; we are just using them in new ways.

Some of these can be written more simply, but for now I want you to see the pattern of subquery testing.

2. Demos with customers and orders and products ordered

These examples will go through similar examples with slightly more complex settings. Remember some of these examples will looks reasonable at first- but may be logically incorrect.

When you create a query you need to consider that the Where clause operates on a single row of the virtual table produced by the from clause.

We want to find customers who have purchased both a stationary bike (product id 1050) and a stationary bike (product id 1060) on the same order. Your first attempt might be the following which is not correct.

Demo 14: Using an In List test

```
select oh.customer_id, order_id, od.prod_id
from orderEntry.orderHeaders oh
join orderEntry.orderDetails od using (order_id)
where prod_id in (1050, 1060)
order by oh.customer_id, order_id, od.prod_id;
+-------------+----------+---------+
| customer_id | order_id | prod_id |
+-------------+----------+---------+
|      401250 |      106 |    1060 |
|      403000 |      536 |    1050 |
|      403000 |     2505 |    1060 |
|      403000 |     4511 |    1060 |
|      404100 |      605 |    1060 |
|      404100 |     2805 |    1060 |
|      404950 |      411 |    1060 |
|      404950 |      535 |    1050 |
|      408770 |      405 |    1050 |
|      408770 |      405 |    1060 |
|      409030 |      128 |    1060 |
|      903000 |      312 |    1050 |
|      903000 |      312 |    1060 |
|      903000 |      312 |    1060 |
+-------------+----------+---------+
14 rows in set (0.08 sec)
```

Looking at the result the first customer returned is customer_id 401250 and that customer ordered product 1060 but not product 1050. We do not want that customer id returned because he did not buy both products.

Demo 15: The In List filter is generally the equivalent of an OR test. Using an OR test

```
select oh.customer_id, order_id, od.prod_id
from orderEntry.orderHeaders oh
join orderEntry.orderDetails od using (order_id)
where prod_id = 1050
or   prod_id = 1060
order by oh.customer_id, order_id, od.prod_id;
```

Demo 16: You might then try an AND operator since you want customers who bought product 1050 AND product 1060. But that returns no rows. It is testing for a purchase of a product that is both 1050 and 1060 on the same order detail row. Think about the output of a join in this query.

```
select oh.customer_id, order_id, od.prod_id
from orderEntry.orderHeaders oh
join orderEntry.orderDetails od using (order_id)
where prod_id = 1050
and   prod_id = 1060
order by oh.customer_id, order_id, od.prod_id;
```

```
Empty Set
```

Demo 17: If you look at the rows in the virtual table created by the FROM clause, we have a series of rows with a single column for the product ID. These are some of those rows.

```
select oh.customer_id, order_id, od.prod_id
from orderEntry.orderHeaders oh
join orderEntry.orderDetails od using (order_id)
order by order_id;
+-------------+----------+---------+
| customer_id | order_id | prod_id |
+-------------+----------+---------+
|      403000 |      105 |    1030 |
|      403000 |      105 |    1020 |
|      403000 |      105 |    1010 |
|      401250 |      106 |    1060 |
|      403050 |      107 |    1110 |
|      403000 |      108 |    1080 |
|      403000 |      109 |    1130 |
|      404950 |      110 |    1090 |
|      404950 |      110 |    1130 |
|      403000 |      111 |    1150 |
|      403000 |      111 |    1141 |
```

Note that each row has one customer_id value and one order_id value and one prod_id value.

With a Where clause each of those rows is checked, one row at a time.

Suppose our Where clause is

```
Where prod_id = 1050 OR prod_id = 1060
```

Then the first row evaluates to False and is not returned. The second row evaluates to False and is not returned.

The third row evaluates to False and is not returned. The fourth row evaluates to True and is returned. **But the test in the Where clause never looks at more than one row at a time.**

Suppose our Where clause is

```
Where prod_id = 1050 AND   prod_id = 1060
```

Then we are looking for rows Where the single value for prod_id in a row is *both* 1050 and 1060. This is never going to happen with our From clause.

But we can solve this problem. We are really asking a question about orders**- so we should start with the order headers table.**

Demo 18: Using two subqueries gives us the correct result. This query says show me the orders where the order has at least one detail row with an order for product 1050 and the (same) order with has at least one detail row with an order for product 1050. The order id has to pass both of these tests to get into the output.

```
select oh.customer_id, order_id
from orderEntry.orderHeaders oh
where order_id in (
   select order_id
   from orderEntry.orderDetails od
   where prod_id = 1050)
and order_id in (
   select order_id
   from orderEntry.orderDetails od
   where prod_id = 1060)
order by oh.customer_id, order_id;
+-------------+----------+
| customer_id | order_id |
+-------------+----------+
|      408770 |      405 |
|      903000 |      312 |
+-------------+----------+
```

We are looking at each row in the order_headers rows and using a Where clause with an AND test. We can read that Where clause as - we want an order id that is in the details table for orders for product 1050 and is also in the details table for orders for product 1060- which is what we want.

If we run just the first subquery:

```
   select order_id
   from orderEntry.orderDetails od
   where prod_id = 1050;
+----------+
| order_id |
+----------+
|      312 |
|      405 |
|      535 |
|      536 |
+----------+
```

If we run the second subquery:

```
   select order_id
   from orderEntry.orderDetails od
   where prod_id = 1060;
+----------+
| order_id |
+----------+
|      106 |
|      128 |
|      312 |
|      312 |
|      405 |
|      411 |
|      605 |
|     2505 |
|     2805 |
|     4511 |
+----------+
```

What we want is the order id values that are in both of those tables.

Now suppose we want to find customers who bought both of these products but not necessarily on the same order. **This is a question about customers.** Now we want to test for the customer id twice- once for an order for product 1050 and once for an order for product 1060.

Demo 19: Using two subqueries gives us the correct result.

```
select customer_id
from customer.customers
where customer_id  in (
   select customer_id
   from orderEntry.orderHeaders oh
   join orderEntry.orderDetails od using (order_id)
   where prod_id = 1050)
and customer_id  in (
   select customer_id
   from orderEntry.orderHeaders oh
   join orderEntry.orderDetails od using (order_id)
   where prod_id = 1060)
order by customer_id;;
+-------------+
| customer_id |
+-------------+
|      403000 |
|      404950 |
|      408770 |
|      903000 |
+-------------+
```

3. The Multi-match pattern

These are not trivial queries. But they are useful queries. The first thing you need to do is recognize this pattern. We want to find customers who purchased product 1050 **and also** product 1060. That is not the same as finding customers who purchased product 1050 or product 1060. The customer has to pass two distinct tests to get into the result.

And it is critical to determine is this is a question that is mainly about customers, or about orders or about departments.

It is easier to recognize patterns if you think of other examples:

1. We want clients at the vet clinic who have both a dog and a cat.
2. We want animals that had an exam last year and also had an exam this year.
3. We want students who passed CS 110A and 110B.
4. We want to hire people who have both SQL Server experience and Oracle experience.

Then you should expand the patterns a bit.

5. We want clients at the vet clinic who have a dog but not a cat. This is still two separate tests.
6. We want animals that had an exam last year but do not have an exam this year. This is a positive match and a negative match.
7. We want students who passed CS 111A and 111B and 111C. This is three matches.
8. We want clients at the vet clinic who have a dog and a cat and a snake. This is three matches.
9. We want students who passed (CS 110A or CS 111A) and 110B.
10. We want clients at the vet clinic who have (a dog or a cat) and a snake.
11. We want clients at the vet clinic who have (a dog or a cat) but not a snake.

The more you do this, the easier it will be to recognize this pattern on the midterm exam and on the final exam. You should also recognize this pattern with respect to targeted advertising and medical decisions (which patients are over 65 years old and do not have a flu shoot- which is rather like example 5.)

Then you have to remember that you cannot solve these problems with a simple table expression in the From clause and a filter that uses an OR test, or an In list- which is an Or test, or a AND test. These are more complex.

The next thing to think through is what you are actually testing. In the last demo, we want to find customers who bought both of these products. So the filter is on the **customer** (customer_id). The tests ask if the **customer** is in the list of people who purchased product 1050 and if the **customer** is in the list of people who purchased product 1060.