Table of Contents

There is another type of query that we will see implemented in various ways over the semester. We are going to start with some examples of single table row filters.

This logic seems to give people some problems, so you need to read through this and think about the logic. It is not difficult but you need to visualize the data produced by the From clause expression and how the Where clause filters works on it.

1. Single table row filters

The tests we have been using in the Where clause are called row filters because they are used to examine the proposed result set one row at a time to see if that row will be allowed in the final result set.

We are going to start with the employee table. This table is set up with the assumption that every 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). This means that for each employee we store only one value for job_id, the employee's current job.

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

```
Select emp_id, dept_id, job_id
From a_emp.employees
Order by dept_id, emp_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 |
+--------+---------+--------+
22 rows in set (0.00 sec)
```

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 a_emp.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 a_emp.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)
```

Before you go to the next demo, think about the previous in more detail and think about how a query is processed.

1) The From clause supplies the data- we have one roe for each row in the employee table.
2) The Where clause examines each row in turn. We cannot determine the order in which the rows are examined but for the sake of argument assume it is in the same order as was return by the first query.

```
Does this row pass the test job_id = 16 or job_id = 32? NO
|    100 |      10 |      1 |

Does this row pass the test job_id = 16 or job_id = 32? NO
|    201 |      20 |      2 |
Does this row pass the test job_id = 16 or job_id = 32? Yes
|    101 |      30 |     16 |

Does this row pass the test job_id = 16 or job_id = 32? Yes
|    108 |      30 |     16 |

Does this row pass the test job_id = 16 or job_id = 32? Yes
|    109 |      30 |     32 |

SEVERAL ROWS SKIPPED
...
Does this row pass the test job_id = 16 or job_id = 32? NO
|    207 |      35 |      8 |

Does this row pass the test job_id = 16 or job_id = 32? NO
|    145 |      80 |      4 |
```

And so on through the rest of the table. That is how a row filter operates. If the row being exmined passes the test is is sent on to the next step in producing the final result set.

Demo 04: If we can 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 have the value 16 and the value 32 for job id at the same time.

```
Select emp_id, dept_id, job_id
From a_emp.employees
Where job_id = 16 AND job_id = 32
Order by emp_id
;
```

```
Empty Set
```

Now think about this Where clause filter and how the data set is being processed

```
Does this row pass the test job_id = 16 AND job_id = 32? NO
|    100 |      10 |      1 |

Does this row pass the test job_id = 16 AND job_id = 32? NO
|    201 |      20 |      2 |

Does this row pass the test job_id = 16 AND job_id = 32? NO
|    101 |      30 |     16 |

Does this row pass the test job_id = 16 AND job_id = 32? NO
|    108 |      30 |     16 |
```

Because this is a row filter and the rows to be examined have a single value for job_id, none of the rows can pass the test for having both the value 16 and the value 32 at the same time.

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

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.

Demo 05:

```
Select distinct dept_id
From a_emp.employees
Where job_id = 16 ;
+---------+
| dept_id |
+---------+
|      30 |
|     215 |
|      35 |
+---------+
```

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

```
Select distinct dept_id
From a_emp.employees
Where job_id = 32 ;
+---------+
| dept_id |
+---------+
|     210 |
|      30 |
|     215 |
+---------+
```

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

```
Select distinct dept_id
From a_emp.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 operator is never met.

Demo 08:

```
Select distinct dept_id
From a_emp.employees
Where job_id = 16  AND job_id = 32
;
```

```
Empty Set
```

But is is reasonable to ask to see departments where we have employees with job 16 and employees with job 32.

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. If this department row meets both tests then it is passed into the result set. 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 set to the previous ones.

```
Select dept_id
From  a_emp.departments
Where dept_id in (
   select dept_id
   From a_emp.employees
   Where job_id =16
   )
and  dept_id in (
   select dept_id
   From a_emp.employees
   Where job_id =32
   );
+---------+
| dept_id |
+---------+
|      30 |
|     215 |
+---------+
```

It might help to think about this by substituting the two subquery results in the query.

```
Select dept_id
From  a_emp.departments
Where dept_id in (
   +---------+
   |      30 |
   |     215 |
   |      35 |
   +---------+
)
and  dept_id in (
   +---------+
   |     210 |
   |      30 |
   |     215 |
   +---------+
);
```

Go through the rows in the department table and test this AND filter against each dept id.

You should be able to think of how to write queries 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 have no employees with job id 32.

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

2. Row filters and sub queries

These examples will go through similar examples with slightly more complex settings. Remember some of these examples will looks reasonable at first- but are 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 10: Using an In List test

```
Select oh.cust_id, ord_id, od.prod_id
From a_oe.order_headers oh
Join a_oe.order_details od using (ord_id)
Where prod_id in (1050, 1060)
Order by oh.cust_id, ord_id, od.prod_id;
+---------+--------+---------+
| cust_id | ord_id | prod_id |
+---------+--------+---------+
|  401250 |    106 |    1060 |
|  403000 |    505 |    1060 |
|  403000 |    511 |    1060 |
|  403000 |    536 |    1050 |
|  404950 |    535 |    1050 |
|  408770 |    405 |    1050 |
|  408770 |    405 |    1060 |
|  409030 |    128 |    1060 |
|  903000 |    312 |    1050 |
|  903000 |    312 |    1060 |
|  903000 |    312 |    1060 |
+---------+--------+---------+
11 rows in set (0.02 sec)
```

Looking at the result set the first customer returned is cust_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. The In List filter is the equivalent of an OR test.

Demo 11: Using an OR test we still get back customer 401250 which is incorrect.

```
Select oh.cust_id, ord_id, od.prod_id
From  a_oe.order_headers oh
Join a_oe.order_details od using (ord_id)
Where prod_id = 1050
or prod_id = 1060
order by oh.cust_id, ord_id, od.prod_id;
+---------+--------+---------+
| cust_id | ord_id | prod_id |
+---------+--------+---------+
|  401250 |    106 |    1060 |
|  403000 |    505 |    1060 |
|  403000 |    511 |    1060 |
|  403000 |    536 |    1050 |
|  404950 |    535 |    1050 |
|  408770 |    405 |    1050 |
|  408770 |    405 |    1060 |
|  409030 |    128 |    1060 |
|  903000 |    312 |    1050 |
|  903000 |    312 |    1060 |
|  903000 |    312 |    1060 |
+---------+--------+---------+
11 rows in set (0.00 sec)
```

Demo 12: You might then try an AND operator since you want customers who bought product 1050 AND product 1060. But that returns no rows.

```
Select oh.cust_id, ord_id, od.prod_id
From  a_oe.order_headers oh
Join a_oe.order_details od using (ord_id)
Where  prod_id = 1050
And prod_id = 1060
Order by oh.cust_id, ord_id, od.prod_id;
```

```
Empty Set
```

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.

Demo 13:

```
Select oh.cust_id, ord_id, od.prod_id
From  a_oe.order_headers oh
Join a_oe.order_details od using (ord_id)
;
+---------+--------+---------+
| cust_id | ord_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 cust_id value and one ord_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.

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.

Demo 14: Using two subqueries gives us the correct result.

```
Select oh.cust_id, ord_id
From a_oe.order_headers oh
Where ord_id in (
   Select ord_id
   From a_oe.order_details od
   Where prod_id = 1050)
And ord_id in (
   Select ord_id
   From a_oe.order_details od
   Where prod_id = 1060)
Order by oh.cust_id, ord_id;
+---------+--------+
| cust_id | ord_id |
+---------+--------+
|  408770 |    405 |
|  903000 |    312 |
+---------+--------+
2 rows in set (0.00 sec)
```

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

If we run just the first subquery:

```
   Select ord_id
   From a_oe.order_details od
   Where prod_id = 1050;
+--------+
| ord_id |
+--------+
|    312 |
|    405 |
|    535 |
|    536 |
+--------+
4 rows in set (0.00 sec)
```

If we run the second subquery:

```
   Select ord_id
   From a_oe.order_details od
   Where prod_id = 1060;
+--------+
| ord_id |
+--------+
|    106 |
|    128 |
|    312 |
|    312 |
|    405 |
|    505 |
|    511 |
+--------+
7 rows in set (0.00 sec)
```

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. 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. We start with the customer table.

Demo 15: Using two subqueries gives us the correct result.

```
Select cust_id
From a_oe.customers
Where cust_id  in (
   Select cust_id
   From a_oe.order_headers oh
   Join a_oe.order_details od using (ord_id)
   Where prod_id = 1050
)
And cust_id  in (
   Select cust_id
   From a_oe.order_headers oh
   Join a_oe.order_details od using (ord_id)
   Where prod_id = 1060
)
Order by cust_id;
+---------+
| cust_id |
+---------+
|  403000 |
|  408770 |
|  903000 |
+---------+
3 rows in set (0.00 sec)
```

These are the customers who purchased product 1050.

```
+---------+
| cust_id |
+---------+
|  903000 |
|  408770 |
|  404950 |
|  403000 |
+---------+
```

These are the customers who purchased product 1060.

```
+---------+
| cust_id |
+---------+
|  401250 |
|  409030 |
|  903000 |
|  903000 |
|  408770 |
|  403000 |
|  403000 |
+---------+
```

These are more complex queries than we have seen before and you do need to spend some time thinking about the logic and not just about the syntax.