Table of Contents

[1. Single table row filters 1](#_Toc363599182)

[2. Row filters and sub queries 4](#_Toc363599183)

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.

1. Single table row filters

The Where clause we have been using is called a row filter because it is used to examine the proposed result set one row 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.

1. 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)

1. 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)

1. 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)

1. 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 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.

Select distinct dept\_id

From a\_emp.employees

Where job\_id = 16 ;

+---------+

| dept\_id |

+---------+

| 30 |

| 215 |

| 35 |

+---------+

1. 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 |

+---------+

1. This 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 test is never met.

Select distinct dept\_id

From a\_emp.employees

Where job\_id = 16 AND job\_id = 32

;

Empty Set

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.

1. 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 |

+---------+

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 no different subqueries than e used in the previous documents; we are just using them in new ways.

1. 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.

1. 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 |

| 403000 | 411 | 1050 |

| 404950 | 535 | 1050 |

| 408770 | 405 | 1050 |

| 408770 | 405 | 1060 |

| 409030 | 128 | 1060 |

| 903000 | 312 | 1050 |

| 903000 | 312 | 1060 |

| 903000 | 312 | 1060 |

+---------+--------+---------+

12 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.

1. Using an OR 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 = 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 |

| 403000 | 411 | 1050 |

| 404950 | 535 | 1050 |

| 408770 | 405 | 1050 |

| 408770 | 405 | 1060 |

| 409030 | 128 | 1060 |

| 903000 | 312 | 1050 |

| 903000 | 312 | 1060 |

| 903000 | 312 | 1060 |

+---------+--------+---------+

12 rows in set (0.00 sec)

1. 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.

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.

1. 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 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 ord\_id

From a\_oe.order\_details od

Where prod\_id = 1050;

+--------+

| ord\_id |

+--------+

| 312 |

| 405 |

| 535 |

| 536 |

+--------+

8 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 |

| 411 |

| 535 |

| 536 |

+--------+

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.

1. 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 |

| 404950 |

| 408770 |

| 903000 |

+---------+

3 rows in set (0.00 sec)

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 set.

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

We want clients at the vet clinic who have both a dog and a cat.

We want animals that had an exam last year and also had an exam this year.

We want students who passed CS 110A and 110B.

We want to hire people who have both SQL Server experience and Oracle experience.

Then you should expand the patterns a bit.

We want clients at the vet clinic who have a dog but not a cat.

We want animals that had an exam last year but do not have an exam this year.

We want students who passed CS 111A and 111B and 111C

We want students who passed (CS 110A or CS 110A) and 110B.

The more you do this, the easier it will be to recognize this pattern on the midterm exam and on the final exam.

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.

Then 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 ( cust\_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.