Table of Contents

1. Row filters

Most of our queries so far have returned all of the data from the table in the From clause. We are working with very small tables. Imagine the output if we had thousands of rows in our tables. We seldom want to see all of the data in a table. Rather we want to see only a subset of the data- a subset that matches some search condition. If you are using the CCSF WebStars system to see your current schedule- you do not want to see everyone's schedule- just yours.

To filter the output add a WHERE clause to the SQL statement after the From clause. The Where clause specifies which rows should be returned. The Where clause contains a logical expression (a predicate or test) that is applied to each row in the table. If the logical expression evaluates to true for a row from the table (if that row passes the test), that row is returned in the result. The row is not returned if the predicate evaluates as False or as Unknown. We will cover a variety of comparison operators and conditional expressions in this unit and in the following units. We will also investigate the concept of Unknown.

These are Row Filters; the test in the Where clause is applied to each row in turn.

Our query model is now

```
   Select col_expressions
   From table_expression
   WHERE predicate
   Order By sort_keys;
```

In this unit, we will look at several filters: we will discuss more filters soon.

- the Is Null and Is Not Null filters
- the In list filter
- the Between filter
- the comparison operators

A few general rules for writing filters

- You cannot test a column alias in a Where clause; you need to test using the column name.
- Character literals are enclosed in single quotes: 'CA', 'Anderson'
- In tests against character literals, leading blanks are significant. The value `'     CA'` does not match the value `'CA'`. Trailing blanks are not significant.

- Numbers are not enclosed in quotes; do not include punctuation such as commas or dollar signs when you write numeric literals
- Dates are more complex; for now write date literals in the standard default format: '2008-04-01'. Note that this is a string literal; MySQL will cast it to a date for date testing.

For these queries, the table expressions will follow the format of databaseName.tableName such as

```
from a_oe.order_headers
```

This means you can run these queries from within any of your databases.

2. Testing for nulls

You may want to write your queries to skip any rows where certain column values are null. The way to test for this is to add a Where clause after the From clause that filters for the nulls.

Testing for missing values requires the use of the IS NULL operator. Think of IS NULL and IS NOT NULL as single operators. You use the same operator IS NULL for any data type that you are testing.

If you try to test using the syntax = NULL, you will get no rows returned. This is not flagged as an error by the dbms but it does not filter for nulls.

Demo 01: Test for empty attributes by using the test IS NULL. You use the same test for attributes of any data type. This tests for a null shipping_mode_id which is a char(6) attribute.

```
select order_id, order_date, sales_rep_id, shipping_mode_id
from orderEntry.orderHeaders
WHERE shipping_mode_id IS NULL;
+----------+---------------------+--------------+------------------+
| order_id | order_date          | sales_rep_id | shipping_mode_id |
+----------+---------------------+--------------+------------------+
|      116 | 2015-11-12 00:00:00 |          155 | NULL             |
|      117 | 2015-11-28 00:00:00 |          150 | NULL             |
|      118 | 2015-11-28 00:00:00 |          150 | NULL             |
|      119 | 2015-11-28 00:00:00 |          155 | NULL             |
|      550 | 2015-08-02 00:00:00 |         NULL | NULL             |
|      551 | 2015-08-03 00:00:00 |         NULL | NULL             |
|     2120 | 2016-01-02 00:00:00 |         NULL | NULL             |
|     2121 | 2016-01-03 00:00:00 |         NULL | NULL             |
+----------+---------------------+--------------+------------------+
8 rows in set (0.00 sec)
```

Demo 02: Test for empty attributes by using the test IS NULL. This is testing an integer column.

```
select order_id, order_date, sales_rep_id
from orderEntry.orderHeaders
WHERE sales_rep_id IS NULL;
+----------+---------------------+--------------+
| order_id | order_date          | sales_rep_id |
+----------+---------------------+--------------+
|      115 | 2015-11-08 00:00:00 |         NULL |
|      525 | 2016-05-09 00:00:00 |         NULL |
|      527 | 2016-05-01 00:00:00 |         NULL |
|      550 | 2015-08-02 00:00:00 |         NULL |
|      551 | 2015-08-03 00:00:00 |         NULL |
|     2120 | 2016-01-02 00:00:00 |         NULL |
|     2121 | 2016-01-03 00:00:00 |         NULL |
|     2225 | 2016-03-09 00:00:00 |         NULL |
|     3227 | 2016-03-01 00:00:00 |         NULL |
+----------+---------------------+--------------+
9 rows in set (0.00 sec)
```

Demo 03: We can use IS NOT NULL to find rows that have a data value.

```
select order_id, order_date, shipping_mode_id
from orderEntry.orderHeaders
WHERE shipping_mode_id IS NOT NULL
order by order_date
limit 5;
+----------+---------------------+------------------+
| order_id | order_date          | shipping_mode_id |
+----------+---------------------+------------------+
|      522 | 2015-04-05 00:00:00 | USPS1            |
|      540 | 2015-06-02 00:00:00 | FEDEX1           |
|      307 | 2015-06-04 00:00:00 | USPS1            |
|      301 | 2015-06-04 00:00:00 | FEDEX2           |
|      302 | 2015-06-04 00:00:00 | USPS1            |
+----------+---------------------+------------------+
```

Demo 04: Note what happens if you use the test = null instead of Is null

```
select order_id, order_date, shipping_mode_id
from orderEntry.orderHeaders
where shipping_mode_id = NULL;

Empty set
```

The null indicator does not equal any value in the table. So testing with = Null or <> Null is not a true test and that filter will return no rows. The use of Nulls in tables is important but null testing is not always obvious.

3. The IN list test

Suppose we want to display all customers named Morris or Morse or Morise. This is testing against a specific set of values and we can use an IN list for this. The list of values is enclosed in parentheses and the values are separated by commas.

3.1. Simple In lists

Demo 05: Using the IN list for text values. Each text value is enclosed in quotes .

```
select customer_id
 , customer_name_last
 , customer_name_first
from customer.customers
where customer_name_last in( 'Morise', 'Morris', 'Morse' ) ;
+-------------+--------------------+---------------------+
| customer_id | customer_name_last | customer_name_first |
+-------------+--------------------+---------------------+
|      401250 | Morse              | Samuel              |
|      402100 | Morise             | William             |
|      404950 | Morris             | William             |
|      408777 | Morise             | Morris              |
|      409010 | Morris             | William             |
+-------------+--------------------+---------------------+
```

Demo 06: Using the IN list for numeric values. It is not an error to have a value in the list which does not match any rows in the table. It is not an error to have a in the list appear more than once.

```
select order_id
 , order_date
 , customer_id
from orderEntry.orderHeaders
where order_id in(101, 107, 95, 125, 107 );
```

```
+----------+---------------------+-------------+
| order_id | order_date          | customer_id |
+----------+---------------------+-------------+
|      107 | 2015-10-02 00:00:00 |      403050 |
|      125 | 2015-12-09 00:00:00 |      409160 |
+----------+---------------------+-------------+
```

Demo 07: You can use the NOT IN test to exclude specified data values.

```
select prod_id, prod_name, catg_id
from product.products
where catg_id NOT IN ('HW', 'PET');
+---------+---------------------+---------+
| prod_id | prod_name           | catg_id |
+---------+---------------------+---------+
|    1010 | Weights             | SPG     |
|    1020 | Dartboard           | SPG     |
|    1030 | Basketball          | SPG     |
|    1040 | Treadmill           | SPG     |
|    1120 | Washer              | APL     |
|    1125 | Dryer               | APL     |
|    1126 | WasherDryer         | APL     |
|    1130 | Mini Freezer        | APL     |
|    2014 | B000005INR          | MUS     |
|    2234 | B000002I7U          | MUS     |
|    2337 | B000005H40          | MUS     |
```

Demo 08: You can use a list that contains only one item.

```
select job_id, job_title, max_salary
from employee.jobs
where max_salary  IN (120000 );
+--------+------------+------------+
| job_id | job_title  | max_salary |
+--------+------------+------------+
|     16 | Programmer |  120000.00 |
+--------+------------+------------+
```

Demo 09: You can use a NOT IN list that contains only one item.

```
select job_id, job_title, max_salary
from employee.jobs
where max_salary NOT IN (120000 );
+--------+---------------+------------+
| job_id | job_title     | max_salary |
+--------+---------------+------------+
|      1 | President     |  100000.00 |
|      2 | Marketing     |   75000.00 |
|      4 | Sales Manager |   60000.00 |
|      8 | Sales Rep     |   30000.00 |
+--------+---------------+------------+
```

The above two queries may look like they should return all rows in one or the other of these queries. But we have 8 rows in the jobs table; one was returned with the IN (12000) test and four with the NOT IN (12000) test. What happened to the other three rows? Neither the IN test nor the NOT IN test return the rows where the max_salary is null. For that you need to test with the IS NULL test.

Demo 10: Test with IS NULL

```
select job_id, job_title, max_salary
from employee.jobs
where max_salary is null
;
```

```
+--------+--------------+-----------+
| job_id | job_title     | max_salary |
+--------+--------------+-----------+
|     32 | Code Debugger |       NULL |
|     64 | DBA           |       NULL |
|    128 | RD            |       NULL |
+--------+--------------+-----------+
```

3.2. In lists that contain nulls

If you try putting a Null in a list you will find that it does not match a row with a null. The rule is that a row is returned if the Where predicate evaluates as True; a null in the table does not match a null in the list. A null value does not match another null value. The logical value of a null matching a null is Unknown; the value of the filter expression must be True for the row to be returned.

Demo 11: Trying to use Null in a list.

```
select job_id, job_title, max_salary
from employee.jobs
where max_salary  IN (120000, null );
+--------+-----------+-----------+
| job_id | job_title  | max_salary |
+--------+-----------+-----------+
|     16 | Programmer |  120000.00 |
+--------+-----------+-----------+
1 row in set (0.01 sec)
```

What may seem more surprising is the following query, which returns no rows. The rule is that a row is returned if the Where predicate evaluates as True; the row is not returned if the predicate evaluates as False or as Unknown. Here we are negating the Unknown value which is still Unknown.

Demo 12: Nulls always get interesting

```
select job_id, job_title, max_salary
from employee.jobs
where max_salary  NOT IN (120000, null );
```

```
Empty set (0.00 sec)
```

3.3. In lists that contain row values

You can test constructed rows with an In test. In MySQL you can use the expression row(30, 101) to refer to a two part value. The word row is optional; I will use it here to emphasis that we are comparing multi-part row values.

Demo 13: Suppose we wanted to find employees in dept 30 with manager 101. We could use the following. The row values is (30, 101) and it is enclosed in parentheses for the In list. The two columns we are comparing are also enclosed in parentheses.

```
select emp_id, name_last, dept_id,  emp_mng
from employee.employees
where row(dept_id,  emp_mng) IN( row(30, 101) );
+--------+----------+---------+---------+
| emp_id | name_last | dept_id | emp_mng |
+--------+----------+---------+---------+
|    108 | Green     |      30 |     101 |
|    203 | Mays      |      30 |     101 |
|    205 | Higgs     |      30 |     101 |
+--------+----------+---------+---------+
```

Demo 14: Now suppose we wanted to find employees in dept 30 with manager 101 and also employees in dept 35 with manager 101.

```
select emp_id
 , name_last
 , dept_id
 , emp_mng
from employee.employees
where row( dept_id, emp_mng ) in (
    row( 30, 101 )
  , row( 35, 101 )
  )
order by dept_id, emp_mng;
+--------+-----------+---------+---------+
| emp_id | name_last | dept_id | emp_mng |
+--------+-----------+---------+---------+
|    108 | Green     |      30 |     101 |
|    203 | Mays      |      30 |     101 |
|    205 | Higgs     |      30 |     101 |
|    162 | Holme     |      35 |     101 |
|    200 | Whale     |      35 |     101 |
+--------+-----------+---------+---------+
```

Demo 15: You could rewrite this without the keyword row- but include the parentheses which make this a row.

```
select emp_id, name_last, dept_id,  emp_mng
from   employee.employees
where (dept_id,  emp_mng) IN( (30, 101), (35, 101) )
order by dept_id, emp_mng;
```

3.4. Testing a literal against an in list

Commonly we think of testing a column against an In list of literals. But with some tests we can reverse that type of thinking. Suppose we are looking for a customer with the name Morise, but we do not know if that is the customer's first or last name. That happens fairly frequently when people fill out forms.

Demo 16: This query looks for the literal Morise in two columns

```
select *
from customer.customers
where 'Morise' IN ( customer_name_first, customer_name_last);
+-------------+--------------------+---------------------+-----------------------+
| customer_id | customer_name_last | customer_name_first | customer_credit_limit |
+-------------+--------------------+---------------------+-----------------------+
|      402100 | Morise             | William             |                   750 |
|      408777 | Morise             | Morris              |                  7500 |
+-------------+--------------------+---------------------+-----------------------+
```

This syntax is not as obvious in meaning as testing a column against a list of values. Avoid this for a simple In list. The following is poor style.

```
select job_id, job_title, max_salary
from employee.jobs
where 120000 IN (max_salary  );
```

4. The BETWEEN test

To test data against a range of values, use BETWEEN. The Between test is an **inclusive** test. If the row being tested matches an end point of the range, the test has a true value, and the row will get into the output display. The range should be an increasing range. If you test `WHERE salary BETWEEN 72000 and 3000` the query will run but no rows will be returned.

Demo 17: Using BETWEEN with a numeric range.

```
select emp_id, name_last AS "Employee", salary
from employee.employees
where salary BETWEEN 65000 AND 70000;
+--------+----------+----------+
| emp_id | Employee | salary   |
+--------+----------+----------+
|    104 | Ernst    | 65000.00 |
|    109 | Fiet     | 65000.00 |
|    160 | Dorna    | 65000.00 |
|    200 | Whale    | 65000.00 |
|    103 | Hunol    | 69000.00 |
+--------+----------+----------+
```

Demo 18: Using NOT BETWEEN to exclude values in the range.

```
select emp_id, name_last as "Employee" , salary
from employee.employees
where salary not between 10000 AND 65000
order by salary;
+--------+----------+-----------+
| emp_id | Employee | salary    |
+--------+----------+-----------+
|    103 | Hunol    |  69000.00 |
|    205 | Higgs    |  75000.00 |
|    146 | Partne   |  88954.00 |
|    206 | Geitz    |  88954.00 |
|    162 | Holme    |  98000.00 |
|    101 | Koch     |  98005.00 |
|    204 | King     |  99090.00 |
|    100 | King     | 100000.00 |
|    161 | Dewal    | 120000.00 |
+--------+----------+-----------+
9 rows in set (0.00 sec)
```

Demo 19: Using BETWEEN with character range.

```
select emp_id, name_last AS "Employee",dept_id
from employee.employees
where name_last BETWEEN 'J' and 'T'
order by name_last;
+--------+----------+---------+
| emp_id | Employee | dept_id |
+--------+----------+---------+
|    100 | King     |      10 |
|    204 | King     |      30 |
|    101 | Koch     |      30 |
|    203 | Mays     |      30 |
|    146 | Partne   |     215 |
|    145 | Russ     |      80 |
|    207 | Russ     |      35 |
+--------+----------+---------+
```

You need to be careful with character range tests. If we had an employee with a last name composed of just the letter T, that employee would be returned. But the employee with the name Tuck is not returned.

Demo 20: Customer with a low credit rating

```
select customer_id as Customer
, customer_name_last as LastName, customer_name_first as FirstName
, customer_credit_limit as CreditLimit
from customer.customers
where customer_credit_limit between 0 and 1000;
+----------+------------+-----------+-------------+
| Customer | LastName   | FirstName | CreditLimit |
+----------+------------+-----------+-------------+
|   400801 | Washington | Geo       |         750 |
|   401250 | Morse      | Samuel    |         750 |
|   402100 | Morise     | William   |         750 |
|   402110 | Coltrane   | John      |         750 |
|   402120 | McCoy      | Tyner     |         750 |
+----------+------------+-----------+-------------+
5 rows in set (0.00 sec)
```

Demo 21: But customers with no credit rating were not returned by the previous query.

```
select customer_id as Customer
, customer_name_last as LastName, customer_name_first as FirstName
, customer_credit_limit as CreditLimit
from customer.customers
where customer_credit_limit is null;
+----------+----------+------------+-------------+
| Customer | LastName | FirstName  | CreditLimit |
+----------+----------+------------+-------------+
|   402500 | Jones    | Elton John |        NULL |
|   405000 | Day      | David      |        NULL |
+----------+----------+------------+-------------+
```

The word "between" is one of those words that can be ambiguous in English, If I ask you how many animals appear between the Cat and the Dog, you will probably say 3.

But if I asked you how many integers there are between 12 and 16 { 12, 13, 14, 15, 16} some people will say 3 (13,14,15) , some will say 5 [12, 13, 14, 15, 16], and a few people will say 4 [12, 13, 14, 15) or (13, 14, 15, 16]. So if someone asks you to write a query that finds values between two points, it is a good idea to ask if they want to include the end points.

The SQL between operator is **always inclusive of the end points.** Your job is to write a query that does the job you were asked to do.

4.1. Gotchas with Between

Some tests to watch out for with the use of Between.

The Between operator will match no rows if the range is descending or if one of the range points is a null.

Demo 22: Range is decreasing

```
select emp_id, name_last AS "Employee", salary
from employee.employees
where salary BETWEEN 70000 AND 65000
;
Empty set (0.00 sec)
```

Demo 23: One end of the range is a null

```
select emp_id, name_last AS "Employee", salary
from employee.employees
where salary BETWEEN 30000 AND null
;
Empty set (0.00 sec)
```

Demo 24: One end of the range is a null

```
select emp_id, name_last AS "Employee", salary
from employee.employees
where salary BETWEEN null AND 51000
;
Empty set (0.00 sec)
```

5. Direct comparison operators

In this section we will look at another major row filtering technique, using the direct comparison operators.

These operators compare two expressions. The SQL comparison operators are;

= > >= < <= != or <>

The two expressions can be of the same type- such as comparing two string values or two integer values. The two expressions can be of types that can be cast to the same type- such as comparing an integer number to a float number. You need to avoid trying to compare two expressions of different types- such as comparing a date value to an integer. In some cases the dbms will attempt to do such comparisons but it is not a good idea.

These demos use the altgeld_mart tables.

5.1. Tests for exact matches

Demo 25: Display only rows with an exact match on Salary.

```
select emp_id
, name_last as "Employee"
, salary
from employee.employees
where salary = 20000;
+--------+----------+----------+
| emp_id | Employee | salary   |
+--------+----------+----------+
|    150 | Tuck     | 20000.00 |
+--------+----------+----------+
```

Demo 26: Some queries do not return any rows. This does not mean the query is incorrect. We just do not have any matching rows. Depending on the client the results might be shown with a header only or just with a message.

```
select emp_id
, name_last as "Employee"
, salary
from employee.employees
where salary = 18888;
```

```
Empty set (0.00 sec)
```

Demo 27: Display only location rows with a country-id of US.

```
select loc_city
, loc_street_address
from employee.locations
where loc_country_id ='US';
+---------------------+---------------------+
| loc_city            | loc_street_address  |
+---------------------+---------------------+
| Southlake           | 2014 Jabberwocky Rd |
| South San Francisco | 2011 Interiors Blvd |
| San Francisco       | 50 Pacific Ave      |
+---------------------+---------------------+
```

Demo 28: MySQL is not case specific on text comparisons.

```
select loc_city
, loc_street_address
from employee.locations
where loc_city ='SAN FRANCISCO';
+---------------+--------------------+
| loc_city      | loc_street_address |
+---------------+--------------------+
| San Francisco | 50 Pacific Ave     |
+---------------+--------------------+
```

Demo 29: Using a Row equality test.

```
select prod_id, prod_name, catg_id, prod_list_price
from product.products
where row(catg_id, prod_list_price ) = row('PET', 2.50)
;
+---------+------------------+---------+-----------------+
| prod_id | prod_name        | catg_id | prod_list_price |
+---------+------------------+---------+-----------------+
|    1142 | Bird seed        | PET     |            2.50 |
|    1143 | Bird seed deluxe | PET     |            2.50 |
+---------+------------------+---------+-----------------+
2 rows in set (0.00 sec)
```

5.2. Tests for non-matches

Demo 30: Use the not equals operator to exclude rows. You can use != or <>

```
select loc_city
, loc_street_address
from employee.locations
where loc_country_id !='US';
```

```
+-------------+-----------------------+
| loc_city    | loc_street_address    |
+-------------+-----------------------+
| Toronto     | 147 Spadina Ave       |
| Munich      | Schwanthalerstr. 7031 |
| Mexico City | Mariano Escobedo 9991 |
+-------------+-----------------------+
```

5.3. Tests for inequalities

Demo 31: Finding jobs with a max salary less than $60,000. Do not include formatting characters- such as the $ or the comma in the literal.

```
select job_id, max_salary
, job_title
from employee.jobs
where max_salary <60000;
+--------+------------+-----------+
| job_id | max_salary | job_title |
+--------+------------+-----------+
|      8 |   30000.00 | Sales Rep |
+--------+------------+-----------+
```

Demo 32: Finding jobs with a max salary greater than or equal to 60000.

```
select job_id, max_salary
, job_title
from employee.jobs
where max_salary >= 60000;
+--------+------------+---------------+
| job_id | max_salary | job_title     |
+--------+------------+---------------+
|      1 |  100000.00 | President     |
|      2 |   75000.00 | Marketing     |
|      4 |   60000.00 | Sales Manager |
|     16 |  120000.00 | Programmer    |
+--------+------------+---------------+
```

6. Tests that require conversions

These are queries that you could try to run that might not work at all in some dbms; that might work with invalid conversions; or that might turn out OK. In any case you should not run these types of queries- care about your data!

Implicit Type casting: Suppose you wrote the Where clause in the first demo as Where salary = '20000'

First of all, you should not do that. The salary attribute is defined as a numeric column and the literal '20000' is a string, not a number. Comparing a numeric attribute to a string is very poor style and makes you look like you do not know how to write code. Most dbms will look at that expression and implicitly cast the string '20000' to a number to do the comparison. But you would need to know all of the cast rules for whatever dbms you are working with and you might occasionally be surprised at the cast that is done. An "implicit" cast is one that is done for you without the system informing you of the cast. So the solution is that you should write the literals correctly and avoid implicit casts when possible.

Demo 33: Comparing a string to a number: You should test the numeric salary attribute against a number- not against a string. You should *not * do this type of test.

```
select emp_id
, name_last as "Employee"
, salary
from employee.employees
where salary = '20000';
+--------+----------+----------+
| emp_id | Employee | salary   |
+--------+----------+----------+
|    150 | Tuck     | 20000.00 |
+--------+----------+----------+
```