Table of Contents

1. 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.

1.1. Tests for exact matches

Demo 01: Display only rows with an exact match on Salary.

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

Demo 02: 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 a_emp.employees
where salary = 18888;
Empty set (0.00 sec)
```

Demo 03: Display only location rows with a country-id of US.

```
select loc_city
, loc_street_address
from a_emp.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 04: MySQL is not case specific on text comparisons.

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

Demo 05: Test date values using the default date format; enclose the date literal in single quotes..

```
select ord_id
, cust_id
, ord_mode
from a_oe.order_headers
where ord_date = '2013-12-15' ;
+--------+---------+----------+
| ord_id | cust_id | ord_mode |
+--------+---------+----------+
|    126 |  409190 | DIRECT   |
|    127 |  915001 | ONLINE   |
|    128 |  409030 | ONLINE   |
|    129 |  915001 | DIRECT   |
+--------+---------+----------+
```

This is one time when we do commonly rely on an implicit cast because '2013-12-15' is actually a string. You could do an explicit cast where ord_date = cast('2013-12-15' as date);

Demo 06: Using a Row equality test.

```
select prod_id, prod_name, catg_id, prod_list_price
from a_prd.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)
```

1.2. Tests for non-matches

Demo 07: Use the not equals operator to exclude rows. You can use != or <>

```
select loc_city
, loc_street_address
from a_emp.locations
where loc_country_id !='US';
+-------------+----------------------+
| loc_city    | loc_street_address   |
+-------------+----------------------+
| Toronto     | 147 Spadina Ave      |
| Munich      | Schwanthalerstr. 7031 |
| Mexico City | Mariano Escobedo 9991 |
+-------------+----------------------+
```

1.3. Tests for inequalities

Demo 08: 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 a_emp.jobs
where max_salary <60000;
+--------+------------+-----------+
| job_id | max_salary | job_title |
+--------+------------+-----------+
|      8 |   30000.00 | Sales Rep |
+--------+------------+-----------+
```

Demo 09: Finding jobs with a max salary greater than or equal to 60000.

```
select job_id, max_salary
, job_title
from a_emp.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    |
+--------+------------+---------------+
```

2. 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 dbmss 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 10: Comparing a string to a number: You should test the numeric salary attribute against a number- not against a string.

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

Demo 11: Demo 05 written with an explicit cast. It is very common to rely on the implicit cast for date literals.

```
select ord_id
, cust_id
, ord_mode
from a_oe.order_headers
where ord_date = cast('2013-12-15' as date);
+--------+---------+----------+
| ord_id | cust_id | ord_mode |
+--------+---------+----------+
|    126 |  409190 | DIRECT   |
|    127 |  915001 | ONLINE   |
|    128 |  409030 | ONLINE   |
|    129 |  915001 | DIRECT   |
+--------+---------+----------+
```

Demo 12: -- comparing a date to a number ; MySQL has a conversion from a number to a date- but it is not obvious. You are better off comparing date attributes to date values..

```
select emp_id
, hire_date
from a_emp.employees
where hire_date> 12345678;
```

This query returns all of the employees we have . We do get a warning.

Warning (Code 1292): Incorrect date value: '12345678' for column 'hire_date' at row 1

What about a different number?

```
select emp_id
, hire_date
from a_emp.employees
where hire_date> 21345678;
Empty set
```

What about a different number?

```
select emp_id
, hire_date
from a_emp.employees
where hire_date> 20130101;
+--------+------------+
| emp_id | hire_date  |
+--------+------------+
|    204 | 2013-06-15 |
|    206 | 2013-06-15 |
+--------+------------+
```

- MySQL automatically converts a date or time value to a number if the value is used in a numeric context and vice versa.
- By default, when MySQL encounters a value for a date or time type that is out of range or otherwise invalid for the type, it converts the value to the “zero” value for that typ