Table of Contents

1. Compound criteria

For more interesting queries, we can use compound criteria. These are criteria that contain multiple conditions joined with the logical operators AND, OR, and NOT.

1.1. The AND logical operator

With this operator, the compound test has a true value if both conditions are true.

Demo 01: We want to see employees hired in 2008.

```
select emp_id, name_last as "Employee", hire_date, salary
from a_emp.employees
where hire_date BETWEEN '2008-01-01' AND '2008-12-31';
+--------+----------+------------+----------+
| emp_id | Employee | hire_date  | salary   |
+--------+----------+------------+----------+
|    101 | Koch     | 2008-06-17 | 98005.00 |
|    145 | Russ     | 2008-03-30 | 59000.00 |
|    205 | Higgs    | 2008-06-01 | 75000.00 |
+--------+----------+------------+----------+
```

Demo 02: We want to see employees hired in 2008 who earn more than 50000. A row has to pass both tests to be included in the result set

```
select emp_id, name_last as "Employee", hire_date, salary
from a_emp.employees
where hire_date BETWEEN '2008-01-01' AND '2008-12-31'
AND salary > 60000;
+--------+----------+------------+----------+
| emp_id | Employee | hire_date  | salary   |
+--------+----------+------------+----------+
|    101 | Koch     | 2008-06-17 | 98005.00 |
|    205 | Higgs    | 2008-06-01 | 75000.00 |
+--------+----------+------------+----------+
```

When we AND in another filter we will generally reduce the number of rows returned by the query.

Demo 03: We want to see jobs that do not seem to be in Sales with a minimum salary more than 40000. We cannot be certain that these are all of the non-sales jobs- just that they are jobs which do not have Sales in the job title.

```
select job_id, min_salary, max_salary
from a_emp.jobs
where job_title NOT LIKE '%Sales%'
AND min_salary > 40000
;
```

```
+--------+------------+------------+
| job_id | min_salary | max_salary |
+--------+------------+------------+
|      1 |  100000.00 |  100000.00 |
|     16 |   60000.00 |  120000.00 |
|     32 |   60000.00 |       NULL |
|     64 |   60000.00 |       NULL |
|    128 |   60000.00 |       NULL |
+--------+------------+------------+
```

Demo 04: This shows employees with a salary between 12000 and 30000

```
select emp_id, name_last as "Employee", salary
from a_emp.employees
where salary between 12000 and 30000
order by salary;
+--------+----------+----------+
| emp_id | Employee | salary   |
+--------+----------+----------+
|    201 | Harts    | 15000.00 |
|    150 | Tuck     | 20000.00 |
|    155 | Hiller   | 29000.00 |
|    207 | Russ     | 30000.00 |
+--------+----------+----------+
```

Demo 05: If you need to exclude the end point, then use fld > x and fld < y.

```
select emp_id, name_last as "Employee", salary
from a_emp.employees
where salary > 12000
AND salary < 30000
Order by salary;
+--------+----------+----------+
| emp_id | Employee | salary   |
+--------+----------+----------+
|    201 | Harts    | 15000.00 |
|    150 | Tuck     | 20000.00 |
|    155 | Hiller   | 29000.00 |
+--------+----------+----------+
```

Demo 06: Avoid writing tests that logically can never have a True value.

```
select emp_id, name_last as "Employee", salary
from a_emp.employees
where salary < 12000
AND   salary > 30000
Order by salary
;
```

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

Demo 07: You are not limited to combining two tests.

```
select emp_id, name_last as "Employee"
, hire_date, salary, job_id
from a_emp.employees
where hire_date between '1985-01-01' and '2005-12-31'
AND  salary > 20000
AND  job_id in (8, 16)
;
```

```
+--------+----------+------------+----------+--------+
| emp_id | Employee | hire_date  | salary   | job_id |
+--------+----------+------------+----------+--------+
|    108 | Green    | 1995-04-14 | 62000.00 |     16 |
|    155 | Hiller   | 2004-03-05 | 29000.00 |      8 |
+--------+----------+------------+----------+--------+
```

Demo 08: Earlier we had a row constructor with an 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);
```

We could do this with an AND test.

```
select prod_id, prod_name, catg_id, prod_list_price
from a_prd.products
where catg_id= 'PET' and prod_list_price = 2.50;
```

1.2. The OR logical operator

With this operator, the compound test has a true value if either one or both conditions are true.

Demo 09: Find employees who work in either dept 20 or 30. It would be better to use an IN operator for this test. Notice that you have to repeat the full test for each OR clause.

```
select emp_id, name_last as "Employee", dept_id
from a_emp.employees
where dept_id = 30
OR    dept_id = 20
order by `Employee`;
+--------+----------+---------+
| emp_id | Employee | dept_id |
+--------+----------+---------+
|    110 | Chen     |      30 |
|    109 | Fiet     |      30 |
|    206 | Geitz    |      30 |
|    108 | Green    |      30 |
|    201 | Harts    |      20 |
|    205 | Higgs    |      30 |
|    204 | King     |      30 |
|    101 | Koch     |      30 |
|    203 | Mays     |      30 |
+--------+----------+---------+
9 rows in set (0.00 sec)
```

Demo 10: Here we want employees who earn more than 70000

```
select emp_id, name_last as "Employee", hire_date, salary, job_id
from a_emp.employees
where salary > 70000;
+--------+------------+-----------+--------+
| emp_id | hire_date  | salary    | job_id |
+--------+------------+-----------+--------+
|    100 | 1989-06-17 | 100000.00 |      1 |
|    101 | 2008-06-17 |  98005.00 |     16 |
|    146 | 2012-02-29 |  88954.00 |     64 |
|    161 | 2011-06-15 | 120000.00 |     16 |
|    162 | 2011-03-17 |  98000.00 |     16 |
|    204 | 2013-06-15 |  99090.00 |     32 |
|    205 | 2008-06-01 |  75000.00 |     16 |
|    206 | 2013-06-15 |  88954.00 |     32 |
+--------+------------+-----------+--------+
8 rows in set (0.00 sec)
```

Demo 11: Here we want employees who earn more than 70000 or who were hired between 1985 and 2005

```
select emp_id, name_last as "Employee", hire_date, salary, job_id
from a_emp.employees
where hire_date between '1985-01-01' and '2005-12-31'
OR    salary > 70000;
+--------+------------+-----------+--------+
| emp_id | hire_date  | salary    | job_id |
+--------+------------+-----------+--------+
|    100 | 1989-06-17 | 100000.00 |      1 |
|    101 | 2008-06-17 |  98005.00 |     16 |
|    108 | 1995-04-14 |  62000.00 |     16 |
|    146 | 2012-02-29 |  88954.00 |     64 |
|    150 | 2001-10-28 |  20000.00 |      8 |
|    155 | 2004-03-05 |  29000.00 |      8 |
|    161 | 2011-06-15 | 120000.00 |     16 |
|    162 | 2011-03-17 |  98000.00 |     16 |
|    201 | 2004-08-25 |  15000.00 |      2 |
|    204 | 2013-06-15 |  99090.00 |     32 |
|    205 | 2008-06-01 |  75000.00 |     16 |
|    206 | 2013-06-15 |  88954.00 |     32 |
+--------+------------+-----------+--------+
12 rows in set (0.00 sec)
```

Demo 12: Now we add another possibility - that the employee's job id is 8 or 16

```
select emp_id, name_last as "Employee" ,hire_date, salary, job_id
from a_emp.employees
where hire_date between '1985-01-01' and '2005-12-31'
OR    salary > 70000
OR    job_id in (8, 16);
+--------+------------+-----------+--------+
| emp_id | hire_date  | salary    | job_id |
+--------+------------+-----------+--------+
|    100 | 1989-06-17 | 100000.00 |      1 |
|    101 | 2008-06-17 |  98005.00 |     16 |
|    108 | 1995-04-14 |  62000.00 |     16 |
|    146 | 2012-02-29 |  88954.00 |     64 |
|    150 | 2001-10-28 |  20000.00 |      8 |
|    155 | 2004-03-05 |  29000.00 |      8 |
|    161 | 2011-06-15 | 120000.00 |     16 |
|    162 | 2011-03-17 |  98000.00 |     16 |
|    200 | 2011-06-17 |  65000.00 |     16 |
|    201 | 2004-08-25 |  15000.00 |      2 |
|    203 | 2010-06-30 |  64450.00 |     16 |
|    204 | 2013-06-15 |  99090.00 |     32 |
|    205 | 2008-06-01 |  75000.00 |     16 |
|    206 | 2013-06-15 |  88954.00 |     32 |
|    207 | 2011-06-17 |  30000.00 |      8 |
+--------+------------+-----------+--------+
15 rows in set (0.00 sec)
```

With each additional Or clause we add, we have the potential of having more rows match.

Demo 13: We had a previous query for max_salary >= 20000 Here we are also including the nulls with an IS NULL test

```
select job_id, job_title, min_salary, max_salary
from a_emp.jobs
where max_salary >= 20000
OR    max_salary is null;
+--------+---------------+------------+------------+
| job_id | job_title     | min_salary | max_salary |
+--------+---------------+------------+------------+
|      1 | President     |  100000.00 |  100000.00 |
|      2 | Marketing     |    5000.00 |   75000.00 |
|      4 | Sales Manager |   15000.00 |   60000.00 |
|      8 | Sales Rep     |   10000.00 |   30000.00 |
|     16 | Programmer    |   60000.00 |  120000.00 |
|     32 | Code Debugger |   60000.00 |       NULL |
|     64 | DBA           |   60000.00 |       NULL |
|    128 | RD            |   60000.00 |       NULL |
+--------+---------------+------------+------------+
```

1.3. The NOT logical operator

The NOT operator works on a single test and reverses the value of that test. The NOT test is often used in combination with AND or OR tests.

Demo 14: We want employees who are **not** in department 20 or 30.

```
select emp_id, name_last as "Employee", dept_id
from a_emp.employees
where NOT dept_id IN ( 30, 20)
order by `Employee`;
+--------+----------+---------+
| emp_id | Employee | dept_id |
+--------+----------+---------+
|    102 | D'Haa    |     215 |
|    161 | Dewal    |     215 |
|    160 | Dorna    |     215 |
|    104 | Ernst    |     210 |
|    155 | Hiller   |      80 |
|    162 | Holme    |      35 |
|    103 | Hunol    |     210 |
|    100 | King     |      10 |
|    146 | Partne   |     215 |
|    207 | Russ     |      35 |
|    145 | Russ     |      80 |
|    150 | Tuck     |      80 |
|    200 | Whale    |      35 |
+--------+----------+---------+
```

The above test could also be written as `where dept_id NOT IN ( 30, 20)` and I think that is easier to read. Note that NOT IN is closer to the way the task is described. I would also encourage you to use `where salary not between 10000 and 20000` instead of `where NOT salary between 10000 and 20000.`

Using the not operator before the tests means that your mind has to keep track of the `NOT` while it reads the rest of the test. Take extra care when using two NOT words in the same test- often people get the logic of double negatives wrong.

1.4. Xor

MySQL supports the XOr operator; this is used when you have two logical expressions and you test that they have different truth values. This is not commonly used but sometimes it is the easiest way to write a query.

Test carefully when you use the XOR operators- most people have trouble with this operator.
The test we are looking at is `dept_id =215  OR   salary > 80000`
compared to the test `dept_id =215  XOR  salary > 80000`

For the first of these tests a row is returned if
   it passes the first component (`dept_id =215`)
or   it passes the second component (`salary > 80000`)
or   it passes both components

For the second of these tests a row is returned if
   it passes the first component (`dept_id =215`) and not the second component (`salary > 80000`)
or   it passes the second component (salary > 80000) and not the first component (`dept_id =215`)

Demo 15: This is a simple OR. Rows are returned if the dept id is 215 or if the salary >80000 or if both are true. We have some rows for people from dept 215 with a salary below 80000

```
select emp_id, name_last as  Employee, dept_id , salary
from a_emp.employees
where dept_id = 215
or salary > 80000
order by dept_id, salary;
+--------+----------+---------+-----------+
| emp_id | Employee | dept_id | salary    |
+--------+----------+---------+-----------+
|    100 | King     |      10 | 100000.00 |
|    206 | Geitz    |      30 |  88954.00 |
|    101 | Koch     |      30 |  98005.00 |
|    204 | King     |      30 |  99090.00 |
|    162 | Holme    |      35 |  98000.00 |
|    102 | D'Haa    |     215 |  60300.00 |
|    160 | Dorna    |     215 |  65000.00 |
|    146 | Partne   |     215 |  88954.00 |
|    161 | Dewal    |     215 | 120000.00 |
+--------+----------+---------+-----------+
```

Demo 16: With the XOR operation a person who is in dept 215 **and** who has a salary >80000 is not returned..

```
select emp_id, name_last as Employee, dept_id , salary
from a_emp.employees
where dept_id =215
Xor  salary > 80000
order by dept_id, salary;
+--------+----------+---------+-----------+
| emp_id | Employee | dept_id | salary    |
+--------+----------+---------+-----------+
|    100 | King     |      10 | 100000.00 |
|    206 | Geitz    |      30 |  88954.00 |
|    101 | Koch     |      30 |  98005.00 |
|    204 | King     |      30 |  99090.00 |
|    162 | Holme    |      35 |  98000.00 |
|    102 | D'Haa    |     215 |  60300.00 |
|    160 | Dorna    |     215 |  65000.00 |
+--------+----------+---------+-----------+
```

2. Hierarchy of evaluation of the logical operators

If you write a criterion that includes more than one logical operator, you need to be concerned about the hierarchy of evaluation. The order of operations is first the NOT operators are evaluated then the ANDs and then the ORs. Parentheses are used to change the order of operations.

Suppose we want to see products that are either pet supplies or sporting goods that cost less than 100. This is an ambiguous statement. Assume this essentially means we want the cheaper sporting good and pet supplies items.

Demo 17: This query following the wording of the task description but does not do the job. We have two Pet items that cost more than $100.

```
select prod_id, prod_list_price, catg_id
from a_prd.products
where catg_id = 'PET' OR catg_id = 'SPG'
AND   prod_list_price < 100;
+---------+-----------------+---------+
| prod_id | prod_list_price | catg_id |
+---------+-----------------+---------+
|    1020 |           12.95 | SPG     |
|    1030 |           29.95 | SPG     |
|    1140 |           14.99 | PET     |
|    1141 |           99.99 | PET     |
|    1142 |            2.50 | PET     |
|    1143 |            2.50 | PET     |
|    1150 |            4.99 | PET     |
|    1151 |           14.99 | PET     |
|    1152 |           55.28 | PET     |
|    4567 |          549.99 | PET     |
|    4568 |          549.99 | PET     |
|    4576 |           29.95 | PET     |
|    4577 |           29.95 | PET     |
+---------+-----------------+---------+
13 rows in set (0.00 sec)
```

Demo 18: If we reverse the testing of the two categories, we get sporting goods items that cost more than $100. That is not right.

```
select  prod_id, prod_list_price, catg_id
from a_prd.products
where catg_id = 'SPG' OR catg_id = 'PET'
AND  prod_list_price < 100;
+---------+-----------------+---------+
| prod_id | prod_list_price | catg_id |
+---------+-----------------+---------+
|    1010 |          150.00 | SPG     |
|    1020 |           12.95 | SPG     |
|    1030 |           29.95 | SPG     |
|    1040 |          349.95 | SPG     |
|    1050 |          269.95 | SPG     |
|    1060 |          255.95 | SPG     |
|    1140 |           14.99 | PET     |
|    1141 |           99.99 | PET     |
|    1142 |            2.50 | PET     |
|    1143 |            2.50 | PET     |
|    1150 |            4.99 | PET     |
|    1151 |           14.99 | PET     |
|    1152 |           55.28 | PET     |
|    4576 |           29.95 | PET     |
|    4577 |           29.95 | PET     |
+---------+-----------------+---------+
15 rows in set (0.00 sec)
```

What is happening here is that we have an AND operator and an OR operator. The rules of precedence is that the AND operator is evaluated first. So the second of these where clauses

```
   where catg_id = 'SPG' or catg_id = 'PET' and prod_list_price < 100;
```

is evaluated as shown here and all of the sporting goods items are returned and Pet supplies that cost more than $100 are returned.

```
   where catg_id = 'SPG' or (catg_id = 'PET' and prod_list_price < 100);
```

We can use parentheses to change the order of evaluation. The order of precedence for these operators is:

NOT

AND

XOR

OR

Demo 19: Adding the parentheses gives us the correct result.

```
   select  prod_id, prod_list_price, catg_id
   from a_prd.products
   where (catg_id = 'SPG' OR catg_id = 'PET')
   AND   prod_list_price < 100;
   +---------+-----------------+---------+
   | prod_id | prod_list_price | catg_id |
   +---------+-----------------+---------+
   |    1020 |           12.95 | SPG     |
   |    1030 |           29.95 | SPG     |
   |    1140 |           14.99 | PET     |
   |    1141 |           99.99 | PET     |
   |    1142 |            2.50 | PET     |
   |    1143 |            2.50 | PET     |
   |    1150 |            4.99 | PET     |
   |    1151 |           14.99 | PET     |
   |    1152 |           55.28 | PET     |
   |    4576 |           29.95 | PET     |
   |    4577 |           29.95 | PET     |
   +---------+-----------------+---------+
   11 rows in set (0.00 sec)
```

Demo 20: It is better to use the IN operator, avoiding the AND/OR Issue.

```
   select  prod_id, prod_list_price, catg_id
   from a_prd.products
   where catg_id IN ( 'SPG', 'PET')
   AND   prod_list_price < 100;
```

3. Rewriting the date test from unit 03

In the discussion of the between test for datetime values we had a query to get exam dates in April 2014. We include a time component in the second range end so that we got exams that had a datetime value sometime during the day of April 30.

```
   select  ex_id
   , stf_id
   , ex_date
   from a_vets.vt_exam_headers
   where ex_date between '2014-04-01' and '2014-04-30 23:59:59';
```

That is somewhat clumsy and also depends on the precision of the time component. We could write this as

```
   select  ex_id
   , stf_id
   , ex_date
   from a_vets.vt_exam_headers
   where ex_date >= '2014-04-01'
   and   ex_date <  '2014-05-01';
```

This is easier to read and understand. It also avoids the use of functions in the Where clause which can make the query more efficient.

4. DeMorgan's laws

Often, there is more than one way to write a complex logical expression. The following equivalencies are known as DeMorgan's Laws.

Where expP and expQ represent logical expressions

NOT (expP AND expQ)	is equivalent to	NOT expP OR NOT expQ
NOT (expP OR expQ)	is equivalent to	NOT expP AND NOT expQ

5. Three-way logic and truth tables

Generally we think of logical expressions having two possible values — True and False. Because database systems allow the use of Null, we have to be concerned with three logical values —True, False, and Unknown. Suppose we have a row in the jobs table with no value for the attribute max_salary , and we evaluate the logical expression: max_salary > 25000 the value of the expression is Unknown for that row. If you are executing a query with a Where clause, if the value of the test is Unknown, the row is not returned.

Remember, NULL is a data value, UNKNOWN is a logical value.

These are the truth tables for the operators NOT, AND, Or and XOR.

The evaluation of the True and False cases are straight forward. With the NOT operator, if I do not know the value of an expression is True or False then I do not know if the negation of that expression is True or False.

NOT	
True	False
Unknown	Unknown
False	True

For the AND operator to Return True both of the operands must have a True value. So if one of the operands is True and the other is unknown, then I cannot know if the ANDed expression is true- so the value is unknown. But if one of the operands is False, then the ANDed expression cannot be true and we know its value is False.

AND	True	Unknown	False
True	True	Unknown	False
Unknown	Unknown	Unknown	False
False	False	False	False

For the OR operator to Return True at least one of the operands must have a True value. So if one of the operands is True and the other is unknown, then the ORed expression is TRUE. If one of the operands is False and the other is unknown then I cannot know the value of the Ored expression and its value is Unknown.

OR	True	Unknown	False
True	True	True	True
Unknown	True	Unknown	Unknown
False	True	Unknown	False

For the XOR operator to Return one of the operands must have a True value and the other operand a False value. If both operands are True or if both operands are False then the result is False.

XOR	True	Unknown	False
True	False	Unknown	True
Unknown	Unknown	Unknown	Unknown
False	True	Unknown	False