**Arithmetic Operators**

You can perform arithmetic operations on numeric fields using arithmetic operators, as tabulated below:

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| DIV | Integer Division |
| % | Modulus (Remainder) |

**Logical Operators - AND, OR, NOT, XOR**

You can combine multiple conditions with boolean operators AND, OR, XOR. You can also invert a condition using operator NOT. For examples,

mysql> SELECT \* FROM products **WHERE quantity >= 5000 AND name LIKE 'Pen %'**;

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

| productID | productCode | name | quantity | price |

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

| 1001 | PEN | Pen Red | 5000 | 1.23 |

| 1002 | PEN | Pen Blue | 8000 | 1.25 |

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

mysql> SELECT \* FROM products **WHERE quantity >= 5000 AND price < 1.24 AND name LIKE 'Pen %'**;

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

| productID | productCode | name | quantity | price |

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

| 1001 | PEN | Pen Red | 5000 | 1.23 |

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

mysql> SELECT \* FROM products **WHERE NOT (quantity >= 5000 AND name LIKE 'Pen %')**;

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

| productID | productCode | name | quantity | price |

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

| 1003 | PEN | Pen Black | 2000 | 1.25 |

| 1004 | PEC | Pencil 2B | 10000 | 0.48 |

| 1005 | PEC | Pencil 2H | 8000 | 0.49 |

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

**IN, NOT IN**

You can select from members of a set with IN (or NOT IN) operator. This is easier and clearer than the equivalent AND-OR expression.

mysql> SELECT \* FROM products **WHERE name IN ('Pen Red', 'Pen Black')**;

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

| productID | productCode | name | quantity | price |

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

| 1001 | PEN | Pen Red | 5000 | 1.23 |

| 1003 | PEN | Pen Black | 2000 | 1.25 |

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

**BETWEEN, NOT BETWEEN**

To check if the value is within a range, you could use BETWEEN ... AND ... operator. Again, this is easier and clearer than the equivalent AND-OR expression.

mysql> SELECT \* FROM products

**WHERE (price BETWEEN 1.0 AND 2.0) AND (quantity BETWEEN 1000 AND 2000)**;

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

| productID | productCode | name | quantity | price |

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

| 1003 | PEN | Pen Black | 2000 | 1.25 |

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

**IS NULL, IS NOT NULL**

NULL is a special value, which represent "no value", "missing value" or "unknown value". You can checking if a column contains NULL by IS NULL or IS NOT NULL. For example,

mysql> **SELECT \* FROM products WHERE productCode IS NULL**;

Empty set (0.00 sec)

Using comparison operator (such as = or <>) to check for NULL is a *mistake* - a very *common mistake*. For example,

SELECT \* FROM products WHERE productCode = NULL;

-- This is a common mistake. NULL cannot be compared.

**ORDER BY Clause**

You can order the rows selected using ORDER BY clause, with the following syntax:

SELECT ... FROM *tableName*

WHERE *criteria*

**ORDER BY *columnA* ASC|DESC, *columnB* ASC|DESC, ...**

The selected row will be ordered according to the values in *columnA*, in either ascending (ASC) (default) or descending (DESC) order. If several rows have the same value in *columnA*, it will be ordered according to *columnB*, and so on. For strings, the ordering could be case-sensitive or case-insensitive, depending on the so-called character collating sequence used. For examples,

-- Order the results by price in descending order

mysql> SELECT \* FROM products WHERE name LIKE 'Pen %' **ORDER BY price DESC**;

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

| productID | productCode | name | quantity | price |

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

| 1002 | PEN | Pen Blue | 8000 | 1.25 |

| 1003 | PEN | Pen Black | 2000 | 1.25 |

| 1001 | PEN | Pen Red | 5000 | 1.23 |

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

-- Order by price in descending order, followed by quantity in ascending (default) order

mysql> SELECT \* FROM products WHERE name LIKE 'Pen %' **ORDER BY price DESC, quantity**;

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

| productID | productCode | name | quantity | price |

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

| 1003 | PEN | Pen Black | 2000 | 1.25 |

| 1002 | PEN | Pen Blue | 8000 | 1.25 |

| 1001 | PEN | Pen Red | 5000 | 1.23 |

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

**AS - Alias**

You could use the keyword AS to define an *alias* for an identifier (such as column name, table name). The alias will be used in displaying the name. It can also be used as reference. For example,

mysql> SELECT **productID AS ID**, **productCode AS Code**,

**name AS Description**, **price AS `Unit Price`**

-- Define aliases to be used as display names

FROM products

**ORDER BY ID**; -- Use alias ID as reference

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

| ID | Code | Description | Unit Price |

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

| 1001 | PEN | Pen Red | 1.23 |

| 1002 | PEN | Pen Blue | 1.25 |

| 1003 | PEN | Pen Black | 1.25 |

| 1004 | PEC | Pencil 2B | 0.48 |

| 1005 | PEC | Pencil 2H | 0.49 |

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

Take note that the identifier "Unit Price" contains a blank and must be *back-quoted*.

##### DISTINCT

A column may have duplicate values, we could use keyword DISTINCT to select only distinct values. We can also apply DISTINCT to several columns to select distinct combinations of these columns. For examples,

-- Without DISTINCT

mysql> **SELECT price FROM products;**

+-------+

| price |

+-------+

| 1.23 |

| 1.25 |

| 1.25 |

| 0.48 |

| 0.49 |

+-------+

-- With DISTINCT on price

mysql> SELECT **DISTINCT price** AS `Distinct Price` FROM products**;**

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

| Distinct Price |

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

| 1.23 |

| 1.25 |

| 0.48 |

| 0.49 |

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

-- DISTINCT combination of price and name

mysql> SELECT **DISTINCT price, name** FROM products**;**

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

| price | name |

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

| 1.23 | Pen Red |

| 1.25 | Pen Blue |

| 1.25 | Pen Black |

| 0.48 | Pencil 2B |

| 0.49 | Pencil 2H |

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

##### GROUP BY Clause

The GROUP BY clause allows you to collapse multiple records with a common value into groups. For example,

mysql> **SELECT \* FROM products ORDER BY productCode, productID;**

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

| productID | productCode | name | quantity | price |

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

| 1004 | PEC | Pencil 2B | 10000 | 0.48 |

| 1005 | PEC | Pencil 2H | 8000 | 0.49 |

| 1001 | PEN | Pen Red | 5000 | 1.23 |

| 1002 | PEN | Pen Blue | 8000 | 1.25 |

| 1003 | PEN | Pen Black | 2000 | 1.25 |

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

mysql> **SELECT \* FROM products GROUP BY productCode;**

-- Only first record in each group is shown

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

| productID | productCode | name | quantity | price |

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

| 1004 | PEC | Pencil 2B | 10000 | 0.48 |

| 1001 | PEN | Pen Red | 5000 | 1.23 |

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

GROUP BY by itself is not meaningful. It is used together with GROUP BY aggregate functions (such as COUNT(), AVG(), SUM()) to produce group summary.

##### GROUP BY Aggregate Functions: COUNT, MAX, MIN, AVG, SUM, STD, GROUP\_CONCAT

We can apply GROUP BY Aggregate functions to each group to produce group summary report.

The function COUNT(\*) returns the rows selected; COUNT(columnName) counts only the non-NULL values of the given column. For example,

-- Function COUNT(\*) returns the number of rows selected

mysql> SELECT COUNT(\*) AS `Count` FROM products**;**

-- All rows without GROUP BY clause

+-------+  
| Count |  
+-------+  
| 5 |  
+-------+

mysql> SELECT productCode, COUNT(\*) FROM products GROUP BY productCode;

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

| productCode | COUNT(\*) |

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

| PEC | 2 |

| PEN | 3 |

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

-- Order by COUNT - need to define an alias to be used as reference

mysql> SELECT productCode, COUNT(\*) AS count

FROM products

GROUP BY productCode

ORDER BY count DESC;

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

| productCode | count |

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

| PEN | 3 |

| PEC | 2 |

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

Besides COUNT(), there are many other GROUP BY aggregate functions such as AVG(), MAX(), MIN() and SUM(). For example,

mysql> SELECT MAX(price), MIN(price), AVG(price), STD(price), SUM(quantity)

FROM products;

-- Without GROUP BY - All rows

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

| MAX(price) | MIN(price) | AVG(price) | STD(price) | SUM(quantity) |

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

| 1.25 | 0.48 | 0.940000 | 0.371591 | 33000 |

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

mysql> SELECT productCode, MAX(price) AS `Highest Price`, MIN(price) AS `Lowest Price`

FROM products

GROUP BY productCode;

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

| productCode | Highest Price | Lowest Price |

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

| PEC | 0.49 | 0.48 |

| PEN | 1.25 | 1.23 |

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

mysql> SELECT productCode, MAX(price), MIN(price),

SUM(quantity)

FROM products

GROUP BY productCode;

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

| productCode | MAX(price) | MIN(price) | SUM(quantity) |

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

| PEC | 0.49 | 0.48 | 18000 |

| PEN | 1.25 | 1.23 | 15000 |

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

##### HAVING clause

HAVING is similar to WHERE, but it can operate on the GROUP BY aggregate functions; whereas WHERE operates only on columns.

mysql> SELECT

productCode AS `Product Code`,

COUNT(\*) AS `Count`,`

FROM products

GROUP BY productCode

HAVING Count >=3;

-- CANNOT use WHERE count >= 3

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

| Product Code | Count |

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

| PEN | 3 |

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