## Table of Contents



We can use another MySQL technique - declaring and using a variable. Most languages have a way to define variables. A variable is a named memory location that can store a value that we can use in our SQL queries. The name/identifier gives us a way to refer to the stored value. To use a variable:

- We need a way to define the variable giving it a name
- We need to consider the data type of the value in the variable
- We need a way to assign a value to that variable and change the value
- We need to consider the scope of the variable- what parts of your code can refer to that variable
- We need to consider the lifetime of the variable- how long does it keep its value

We will limit this discussion to assigning a value to a user-defined variable and then using that variable in an SQL statement.

The variables we are considering here are scalar variables- which means they hold a single data value.

# 1. User-defined variable demo

## 1.1. Define a variable, assign a value, and display

We will start with a demo of a variable and then discuss what it is and why you might want to use these.

Demo 01: define and display variable

```
use a_prd;
set @v_price = 12.95;

select @v_price as Price;
+-------+
| Price |
+-------+
| 12.95 |
+-------+
```

The commands run without error; we are creating a variable, giving it a numeric value and then displaying it with a select query.

## 1.1. Using the variable in a row filter

Demo 02: Now try the following SQL statement which uses that variable in a row filter. I am changing the value of the variable.

```
set @v_price = 29.95;
```

```
Select prod_id, prod_list_price, catg_id
From a_prd.products
Where prod_list_price = @v_price;
+---------+-----------------+---------+
| prod_id | prod_list_price | catg_id |
+---------+-----------------+---------+
|    1030 |           29.95 | SPG     |
|    4576 |           29.95 | PET     |
|    4577 |           29.95 | PET     |
+---------+-----------------+---------+
3 rows in set (0.00 sec)
```

Demo 03: Now we will change the value of the variable and run the query again. The Select statements are the same and use the current value of the variable.

```
set @v_price = 25.50;

Select prod_id, prod_list_price, catg_id
From a_prd.products
Where prod_list_price = @v_price;
+---------+-----------------+---------+
| prod_id | prod_list_price | catg_id |
+---------+-----------------+---------+
|    1070 |           25.50 | HW      |
|    1071 |           25.50 | HW      |
|    1072 |           25.50 | HW      |
+---------+-----------------+---------+
```

Demo 04: We could set a variable for a tax rate and then use it in a calculation

```
set @sales_tax_rate := 0.095;

Select prod_id
, quantity_ordered as Quantity
, quoted_price as Price
, quantity_ordered * quoted_price as AmtDue
, quantity_ordered * quoted_price * @sales_tax_rate as SalesTaxDue
From a_oe.order_details
Limit 5;
+---------+----------+--------+--------+-----------------------------------+
| prod_id | Quantity | Price  | AmtDue | SalesTaxDue                       |
+---------+----------+--------+--------+-----------------------------------+
|    1030 |       12 |  25.00 | 300.00 | 28.500000000000000000000000000000 |
|    1020 |       12 |  12.95 | 155.40 | 14.763000000000000000000000000000 |
|    1010 |        5 | 150.00 | 750.00 | 71.250000000000000000000000000000 |
|    1060 |        1 | 255.95 | 255.95 | 24.315250000000000000000000000000 |
|    1110 |        1 |  49.99 |  49.99 |  4.749050000000000000000000000000 |
+---------+----------+--------+--------+-----------------------------------+
```

# 2. What is happening and some syntax issues

A variable is a named place in memory that stores a value.

## 2.1. Set

The Set statement is used to define and initialize the variable; you can use either the = symbol or the := symbol to do the assignment.

```
Set @ID = 45;
Set @Name := 'Jones';
```

Note that there is no separate declaration statement for the variable. You define a variable by referring to it.

## 2.2. Identifiers

The variable names start with an @ character. You can use letters, digits and underscores in a variable name. The variable names are case neutral. The following uses the two variables defined above.

```
select @ID, @Name, @NAME;
+------+-------+-------+
| @ID  | @Name | @NAME |
+------+-------+-------+
| 45   | Jones | Jones |
+------+-------+-------+
```

## 2.3. Data type

The variable is not given a data type; it picks up the data type from the assigned value. You can assign a value of one data type to a variable and then redefine the variable with a value of a different data type. This attitude is more common in scripting languages so traditional programmers might not expect this. It is better to avoid coding styles that are confusing.

The variable can take a value of integer, decimal, floating-point, binary or nonbinary(character) string.

Demo 05: Using a string variable

```
set @target := 'Shingler Hammer';

Select prod_id, prod_list_price, prod_name
From a_prd.products
Where prod_name  = @target;
+---------+-----------------+-----------------+
| prod_id | prod_list_price | prod_name       |
+---------+-----------------+-----------------+
|    5005 |           45.00 | Shingler Hammer |
+---------+-----------------+-----------------+
```

Demo 06: The value can be set as an expression

```
set @r := 25/8 + 4 + 4 * 1.0;
Select @r;
+--------------+
| @r           |
+--------------+
| 11.125000000 |
+--------------+
```

## 2.4. Unassigned Variables

If you do not assign a value to a variable you can still use the variable; it will have a null string value.

```
Select @NewOne;
+---------+
| @NewOne |
+---------+
| NULL    |
+---------+
```

### 2.5. Using variables to define another variable

Demo 07: You can use one variable to define another variable.

```
set @r := 25/8 + 4 + 4 * 1.0;
Set @r2 := @r + 3;
Select @r2;
+--------------+
| @r           |
+--------------+
| 14.125000000 |
+--------------+
```

Demo 08: This builds up a variable to use with a Like operator.

```
set @target := 'Hammer';
set @target2 := concat('%', @target, '%');
Select  prod_id, prod_list_price, prod_name
From a_prd.products
Where prod_name Like @target2;
+---------+-----------------+------------------+
| prod_id | prod_list_price | prod_name        |
+---------+-----------------+------------------+
|    5002 |           23.00 | Ball-Peen Hammer |
|    5004 |           15.00 | Dead Blow Hammer |
|    5005 |           45.00 | Shingler Hammer  |
+---------+-----------------+------------------+
```

### 2.6. Using the variable, scope, lifespan

You can use the variables in most places where the expression would be valid. But you cannot use a variable to supply the table or column name; and you cannot use a variable to supply the value for a Limit clause.

Here we are assigning values to the variables; commonly the values for these variables would come from the application level programs.

The scope of the user-defined variable is the session you are logged in. If you open another connection to mysql, the value of the variable @numProd will be null in the new session because it is a separate variable. A variable defined in one client session cannot be seen by another client session.

The lifespan of the user-defined variable is the session you are logged in. When you disconnect your session the user-variable is no longer defined. This also means that is your connection is broken and then reconnected, the user-variables have lost their values.

If there is a need to keep the values of these variables, you should store them in a table created for that purpose.

## 3. Some advantages in using variables

One advantage in the use of variables in your queries is when you need to test your queries for different values for the filter tests.

1) Suppose you have a long complex query that need to test the product category value several times. If you store the value for the product category in a variable, then you can change that variable one time and have all references to it in the query change. If you wrote the literal of the category in the query, then you have to find and change each of those values

.

2) Suppose you had a query that needs to filter on the list price of an item and also filter on the quantity of the item. And your first test was

```
Where prod_list_price > 50 and quantity > 50;
```

If you then need to change the filter for quantity to 60 ( and remember you have this test several times in your query) you have to be certain to change only the quantity test and not the price test.

Setting up two varaiables and testing

```
Where prod_list_price > @priceLimit and quantity > @quantityLimit
```

3) Another example you will have in assignments is a filter that tests a date value compared to the current date. We have not discussed date expressions to any detail yet, but suppose you have this test. We want to find orders with the same month number as the current month. So if we run this query in August, it will find orders placed in August, ignoring the month and year.

```
Where month(order_date) = month(current_date())
```

How do you test that query for validity if you need to be able to run in at a later time (remember the actual test may be much more complex).

Suppose you set up a variable

```
Set @dtm = current_date();
```

And use that in your query

```
Where month(ord_date) = month(@dtm)
```

Now you can change the value of the variable to other date values and run the same query using a different "current date".

4) Using variables can help you think more methodically about your queries. With the previous example you could write:

```
Set @dtm = current_date();
Set @curr_month = month(@dtm);
```

and then use the following filter which is closer to the wording of the task.

```
Where month(ord_date) = @curr_month
```

You can also display the value of @curr_month to check that part of the calculation.