Table of Contents

[1. User-defined variable demo 1](#_Toc363597580)

[1.1. Define a variable, assign a value, and display 1](#_Toc363597581)

[1.1. Using the variable in a row filter 1](#_Toc363597582)

[2. What is happening and some syntax issues 2](#_Toc363597583)

[2.1. Set 2](#_Toc363597584)

[2.2. Identifiers 3](#_Toc363597585)

[2.3. Data type 3](#_Toc363597586)

[2.4. Unassigned Variables 3](#_Toc363597587)

[2.5. Using variables to define another variable 3](#_Toc363597588)

[2.6. Using the variable, scope, lifespan 4](#_Toc363597589)

[3. Some advantages in using variables 4](#_Toc363597590)

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

* 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 called scalar variables- which means they hold a single data value.

1. User-defined variable demo

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.

1. 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. Using the variable in a row filter

1. Now try the following SQL statement which uses that variable in a row filter.

set @v\_price = 12.95;

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 |

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

| 1020 | 12.95 | SPG |

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

1. 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 |

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

1. 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 |

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

1. What is happening and some syntax issues

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

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. You define a variable by referring to it.

Identifiers

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

select @ID, @Name, @NAME;

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

| @ID | @Name | @NAME |

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

| 45 | Jones | Jones |

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

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.

1. 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 |

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

1. The value can be set as an expression

set @r := 25/8 + 4 + 4 \* 1.0;

Select @r;

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

| @r |

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

| 11.125000000 |

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

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 |

+---------+

Using variables to define another variable

1. 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 |

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

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 |

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

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.

1. 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 set 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

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(order\_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".