Advanced MySQL user variable techniques

Fri, Dec 15, 2006 in Databases

MySQL's user variables have interesting properties that enable the useful techniques I wrote about in recent articles. One property is that you can read from and assign to a user variable simultaneously, because an assignment can be an r-value (the result of the assignment is the final value of the variable). Another property, which sometimes causes confusing behavior, is un-intuitive evaluation time. In this post I'll show you how to make sure your variables get updated at the time they're used, instead of potentially reading and updating them at different stages of query execution. This technique enables a whole new range of applications for user variables. As a bonus, it also avoids extra columns of output created by variable manipulations.

I will cover several things in this article: assignments as r-values and its side effects, lazy evaluation and its side effects, and finally a technique that lets you have non-lazy evaluation and avoid some side effects.

Setup

I'll use the same data as in recent articles:

```
CREATE TABLE fruits (
  type varchar(10) NOT NULL,
  variety varchar(20) NOT NULL,
  price decimal(5,2) NOT NULL default 0,
  PRIMARY KEY  (type,variety)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

insert into fruits(type, variety, price) values
('apple',  'gala',       2.79),
('apple',  'fuji',       0.24),
('apple',  'limbertwig', 2.87),
```

```
('orange', 'valencia',   3.59),
('orange', 'navel',      9.36),
('pear',   'bradford',   6.05),
('pear',   'bartlett',   2.14),
('cherry', 'bing',       2.55),
('cherry', 'chelan',     6.33);
```

Simultaneous assignment and reading

MySQL lets you assign and read a variable at the same time. This is familiar in many programming languages where an assignment can be an r-value. For example,

```
set @test1 := 0;
set @test2 := @test1 := 5;
select @test1, @test2;
+--------+--------+
| @test1 | @test2 |
+--------+--------+
| 5      | 5      |
+--------+--------+
```

The second statement sets `@test1` to 5, and then sets `@test2` to the result of that assignment, which is the current value of `@test1` . My previous articles have shown you how to exploit this to number rows in a result set, among other things. For example, you can keep a running count as MySQL processes rows, updating and returning the count at the same time.

Side effects

Unfortunately, it got a bit messy sometimes. For example, the following batch, which restarts the numbering every time `type` changes, spews an extra `dummy` column into the output, because that column is where the calculations are taking place:

```
set @type := '', @num := 1;

select type, variety,
   @num := if(@type = type, @num + 1, 1) as row_n
umber,
   @type := type as dummy
from fruits
order by type, variety;
```

```
+--------+------------+------------+--------+
| type   | variety    | row_number | dummy  |
+--------+------------+------------+--------+
| apple  | fuji       |          1 | apple  |
| apple  | gala       |          2 | apple  |
| apple  | limbertwig |          3 | apple  |
| cherry | bing       |          1 | cherry |
| cherry | chelan     |          2 | cherry |
| orange | navel      |          1 | orange |
| orange | valencia   |          2 | orange |
| pear   | bartlett   |          1 | pear   |
| pear   | bradford   |          2 | pear   |
+--------+------------+------------+--------+
```

In previous articles I suggested wrapping that query in a subquery so you can pick which columns you want in the output. That is a bit inefficient (it creates a temporary table internally) and feels kind of kludgey.

Lazy evaluation

MySQL doesn't evaluate expressions containing user variables until they are sent to the client, so some expressions don't work as expected. Setting a variable in one place (such as the `SELECT` list) and reading it another (such as the `HAVING` clause) might give weird results, like as those I demonstrated in my last article where every row was numbered 1 instead of getting incremented as expected.

Here's further clarification from the manual:

> In a SELECT statement, each expression is evaluated only when sent to the client. This means that in a HAVING, GROUP BY, or ORDER BY clause, you cannot refer to an expression that involves variables that are set in the SELECT list. For example, the following statement does not work as expected:
>
> ```
> mysql> SELECT (@aa:=id) AS a, (@aa+3)
> AS b FROM tbl_name HAVING b=5;
> ```
>
> The reference to b in the HAVING clause

> refers to an alias for an expression in the SELECT list that uses @aa. This does not work as expected: @aa contains the value of id from the previous selected row, not from the current row.

In other words, the "alias" in the `HAVING` clause is probably a pointer to a memory location, whose content is not determined for the current row until the current row is output to the client – at which point it's too late to apply any `HAVING` criteria to the row.

Side effects of lazy evaluation

In my last article I showed you how to select the top N rows from each group with user variables. To make that work right, I had to group the query, use a `HAVING` clause, and force a certain index order for that query – because of lazy evaluation. Otherwise, I might have been able to just use the variable in a `WHERE` clause, right? Lazy evaluation is why this doesn't work:

```
set @type := '', @num := 1;

select type, variety, price,
       @num := if(@type = type, @num + 1, 1) as r
ow_number,
       @type := type as dummy
from fruits
where @num <= 2;

+-------+-----------+-------+-----------+------
-+
| type  | variety   | price | row_number | dummy
|
+-------+-----------+-------+-----------+------
-+
| apple | gala      |  2.79 |          1 | apple
|
| apple | fuji      |  0.24 |          2 | apple
|
| apple | limbertwig |  2.87 |          3 | apple
|
+-------+-----------+-------+-----------+------
-+
```

That last row gets output even though it seems `@num`

should have the value 3, eliminating it from the results. However, you can infer from this behavior that `@num` really had the value 2 at the time the `WHERE` clause was evaluated, and was only incremented to 3 after the row was sent to the client.

This aspect of user variable behavior makes user variables significantly harder to understand. Sometimes the results are non-deterministic and/or hard to predict. It would be great if there were a way to update those variables in the context in which they're declared, so they get assigned and read at the same time, instead of having to wait for rows to be sent to the client – a different step in the query execution plan.

Forcing variable evaluation with multi-staged queries

If you understand the order of the steps MySQL uses to execute a query, you can see there are opportunities to make MySQL "finish up" variable assignments before sending the query to the next step. In fact, perhaps it's a bit misleading to say assignments in the `SELECT` are done when rows are sent. I think it's more accurate to say they're done when rows are *generated* for each stage in query execution.

You can see this in a subquery in the `FROM` clause, which is internally stored as an intermediate temporary table. Variable assignments are done before or as the rows are stored in the temporary table, so when results are read from the temporary table, there are no funny

Let me show you the previous query slightly rewritten, and you'll see what I mean:

```
set @type := '', @num := 1;

select type, variety, price, row_number
from (
   select type, variety, price,
       @num := if(@type = type, @num + 1, 1) as r
ow_number,
       @type := type as dummy
```

```
   from fruits
) as x
where row_number <= 2;

+--------+----------+-------+------------+
| type   | variety  | price | row_number |
+--------+----------+-------+------------+
| apple  | gala     |  2.79 |          1 |
| apple  | fuji     |  0.24 |          2 |
| orange | valencia |  3.59 |          1 |
| orange | navel    |  9.36 |          2 |
| pear   | bradford |  6.05 |          1 |
| pear   | bartlett |  2.14 |          2 |
| cherry | bing     |  2.55 |          1 |
| cherry | chelan   |  6.33 |          2 |
+--------+----------+-------+------------+
```

Just by introducing an intermediate step in the query, I forced the variables to be evaluated so the results, when they get to the outer `WHERE` clause, are deterministic. But as I mentioned before, this is kind of kludgey, and depending on the data, it might not be very efficient to create an intermediate temporary table for the results.

Are there better ways? You bet!

Try 1: Use functions to force immediate evaluation

Here's an idea: what if certain functions evaluate their arguments immediately? You could exploit that to create a context that has to be evaluated first, sort of like parenthesizing an expression in an equation. You know, `a = (a + b) * (b + c)` means "do the additions first," which wouldn't be the case without the parentheses – normally multiplication comes before addition.

For this to work, you'd need a function that guarantees the expression is evaluated. For example, `COALESCE()` might be a good choice as long as you put the expression first in the argument list, since `COALESCE()` shortcuts and doesn't evaluate any more arguments as soon as it find a non-NULL argument.

Theoretically, then you could write something like the following and get the desired results:

```
set @type := '', @num := 1;

select type, variety, price,
   coalesce(@num := if(@type = type, @num + 1,
1)) as row_number
...
```

It doesn't work. Why not? Because the `COALESCE` itself isn't evaluated until the rows are generated. So much for that idea.

What about a scalar subquery, then?

```
set @num := 0, @type := '';

select type, variety, price,
   (select(@num := if(@type = type, @num + 1,
1))) as row_number,
...
```

Sorry, no dice. This gives exactly the same results.

This idea will not work, period. *Each and every expression in the `SELECT` list is evaluated as the rows are generated.* Functions are expressions, scalar subqueries are expressions... the only things that will work are operations that result in rows being evaluated for a final value.

Try 2: Force my will on the query

One thing I do know: subqueries in the `FROM` clause are materialized to a temp table, so this will definitely result in rows being generated. This might do what I want, at the expense of generating temporary tables willy-nilly:

```
set @type := '', @num := 1;

select type, variety, price,
   (select n from (select @num := if(@type = typ
e, @num + 1, 1) as n) as x) as row_number,
   (select t from (select @type := type as t) as
x) as dummy
from fruits
where @num <= 2;
```

That won't work either, as it turns out. The subqueries are correlated – they refer to columns from the outer table. That isn't allowed because of the intermediate step, which insulates the inner queries from the outer. This is a limitation of correlated subqueries: you can't nest a subquery in the `FROM` clause inside them.

This is really getting silly. It's time to stop trying to force this to work.

Try 3: Work with me, son

What if I stop trying to get the `SELECT` clause to be evaluated at the same time as the `WHERE` clause? What if I work *with* the server's order of operations, and do all the evaluating *and* updating in the `WHERE` clause instead of in two places? Maybe it looks like this:

```
set @num := 0, @type := '';

select type, variety, price, @num
from fruits
where
   2 >= @num := if(@type = type, @num + 1, 1)
   and @type := type;

+--------+-----------+-------+------+
| type   | variety   | price | @num |
+--------+-----------+-------+------+
| apple  | gala      |  2.79 | 0    |
| apple  | fuji      |  0.24 | 0    |
| apple  | limbertwig |  2.87 | 0    |
| orange | valencia  |  3.59 | 0    |
| orange | navel     |  9.36 | 0    |
| pear   | bradford  |  6.05 | 0    |
| pear   | bartlett  |  2.14 | 0    |
| cherry | bing      |  2.55 | 0    |
| cherry | chelan    |  6.33 | 0    |
+--------+-----------+-------+------+
```

Hmm, that was not really what I wanted. It looks like the variable is never getting updated at all! I'm not sure why not. Maybe if I 'parenthesize' the variable expression like I tried before? I'll use the `GREATEST()` function, which I know will evaluate all its arguments instead of short-cutting like `COALESCE()`:

```
set @num := 0, @type := '';

select type, variety, price, @num
from fruits
where
   2 >= @num := greatest(0, if(@type = type, @num
+ 1, 1))
   and @type := type;
```

No, that gives the same result. I feel like I'm getting close, though. What if I separate out the assignment and comparison?

```
set @num := 0, @type := '';

select * from fruits
where @num := if(type = @type, @num + 1, 1)
   and @type := type
   and @num <= 2;
Empty set (0.00 sec)

select @num, @type;
+------+-------+
| @num | @type |
+------+-------+
| 0    | 0     |
+------+-------+
```

That didn't work either. How did `@type` get assigned an integer? It should be a string. It turns out the `:=` operator has the lowest possible operator precedence, so that `WHERE` clause is actually equivalent to

```
where @num := (
   if(type = @type, @num + 1, 1)
      and (@type := (
         type and @num <= 2)));
```

If I use parentheses right, maybe I can get it to do what I want:

```
select * from fruits
where (@num := if(type = @type, @num + 1, 1))
      and (@type := type)
      and (@num <= 2);
Empty set (0.00 sec)
```

```
select @num, @type;
+------+--------+
| @num | @type  |
+------+--------+
| 9    | cherry |
+------+--------+
```

Now I've gotten the variables to be assigned, but the `WHERE` clause is still eliminating all the rows. This feels so close to being right. What's missing?

Pay dirt: do the assignment inside the function

In fact, I was very close. All I need to do is move the entire assignment and the evaluation inside the function. It seems the variable expressions need to be sealed away from the comparison operator. In the example below, I've put everything inside the `GREATEST()` function, but the expression that updates `@type` has an incompatible type (string), so I convert it to a number with `LENGTH()` and mask its value with `LEAST()` .

```
set @num := 0, @type := '';

select type, variety, price, @num
from fruits
where 2 >= greatest(
   @num := if(@type = type, @num + 1, 1),
   least(0, length(@type := type)));
```

The entire `GREATEST()` expression evaluates to the resulting value of `@num` , which is what I want on the right-hand side of the comparison. And guess what? This works:

```
+--------+----------+-------+------+
| type   | variety  | price | @num |
+--------+----------+-------+------+
| apple  | gala     |  2.79 | 1    |
| apple  | fuji     |  0.24 | 2    |
| orange | valencia |  3.59 | 1    |
| orange | navel    |  9.36 | 2    |
| pear   | bradford |  6.05 | 1    |
| pear   | bartlett |  2.14 | 2    |
| cherry | bing     |  2.55 | 1    |
| cherry | chelan   |  6.33 | 2    |
+--------+----------+-------+------+
```

After playing with more and more combinations, I found another way that works too:

```
select *, @num
from fruits
where
   (@num := if(type = @type, @num + 1, 1)) is not
null
   and (@type := type) is not null
   and (@num <= 2);
```

I confess, I don't fully understand this. I figured it out through trial and error. If the user manual explains it well enough for me to have gotten there by reason, I don't know where. Can someone make it make sense please? I don't want to have to read the source…

What's so great about this?

Two words: one pass. One pass through the table – no quadratic-time algorithms, no grouping or sorting. This is highly efficient. I showed you another technique with `UNION` in my last article, which might be more efficient in some cases. But if you have lots of types of fruit, each of which has just a few varieties, you will be hard-pressed to find a more efficient algorithm to output the first two rows from each group. In fact, I doubt it can be done.

Spurious columns are gone

Putting the variable assignments inside functions not only let me put everything into the `WHERE` clause, it also got rid of the extra columns in the output – without kludges like subqueries. You can use this technique to clean up your output whenever you're doing row-by-row calculations.

Notice the order of rows!

As in previous articles, rows are processed and numbered in order. I never really stated what I was trying to accomplish in the example above. The query I showed you will just output a maximum of two consecutive rows of the

same type, in the order they're read from the table (actually, I guess that's the order they pass through the `WHERE` filter, which might not be the same). If I want to do something specific, such as get the two cheapest varieties from each type of fruit, I need to add an explicit `ORDER BY` to get the rows in order of price:

```
set @num := 0, @type := '';

select type, variety, price, @num
from fruits
where 2 >= greatest(
   @num := if(@type = type, @num + 1, 1),
   least(0, length(@type := type)))
order by type, price;
```

Exercise for the reader: run this query without an index that can be used for ordering. What's in the `@num` column? Why? Add an index on `(type, price)` and try again. How does it change? Why? `EXPLAIN` the queries to find out.

Is that all?

Nope. If you can put user-variable evaluations inside a function, you can put them anywhere you can put a function. That means you could, for example, put them in the `ORDER BY` clause, in the `JOIN` clause, in the `HAVING` clause… anywhere. Now that you know you can do this, you can manipulate variables in lots of places you couldn't do otherwise.

Conclusion

In this article I showed you how two properties of MySQL's user variables (assignment is an r-value, and lazy evaluation) simultaneously cause side effects and give you great power. I showed you why you simply can't get around the fact that the `WHERE` clause and the `SELECT` list are evaluated at different times (I proved it by figuratively banging my head against a wall). I then showed you how you can tuck variable manipulations inside functions, masking out the manipulations and just

getting the result, which can be used in a `WHERE` clause or anywhere else. You now have the tools you need to avoid the side effects of those properties I mentioned.

Finally, I showed you one place you might want to use such a technique to get the first N rows from each group. In certain cases, I think this is the most efficient algorithm possible, requiring just one pass through the table.

I don't know about you, but this opens up a lot of interesting possibilities. I have one particular use in mind that I'll write about next – another way to linearize a query that's normally extremely expensive.

What do you think? Leave a comment and let me know!

Note: I'm taking a break from computers. This is pre-recorded. I'll moderate your comments shortly.

I'm Baron Schwartz, the founder and CEO of VividCortex. I am the author of High Performance MySQL and many open-source tools for performance analysis, monitoring, and system administration. I contribute to various database communities such as Oracle, PostgreSQL, Redis and MongoDB.

Newer | Older

Comments

Andrea · 9 years ago

Hi.

So many things to tell. Let's see.

I love your posts. I discovered your site a couple of days ago, and I browsed all till September 2005, also reading many articles. Thanks for posting. I was so impressed that I restructured the links on my NotesLog to properly point to your site.

I found you while looking for info about a problem in WordPress. Not a bug, but a weak implementation. Which is also simple and flexible, though: database access is not structured. Anyone can write any query and change and retrieve raw data.

I was writing a plugin for hiding categories together with their posts, based on the current user's roles. Leaving out the details, I got to the conclusion that

see more

Reply · Share ›

Xaprb · 9 years ago

Hi Andrea, thanks for the kind words. As for a parser, you could possibly use MySQL's own, either by starting from the source code (sql/sql_yacc.yy and friends) or starting from a similar project in Perl, which I believe hooks into the MySQL libraries and gets them to do the work (DBIx::MyParse).

I suspect there is an easier way to accomplish it, or maybe you could find a way to change your requirements to something easier to do though. The approach you mention seems likely to be very hard to get correct.

Reply · Share ›

Andrea · 9 years ago

I saw that library by Philip Stoev before, and I think it's what I was looking for, but it has at least two drawbacks: 1- it's perl, not php; 2- it's only one month ago that Philip was looking for answers from MySQL developers on how to make it work... But, yes, I think I'm going to contact him anyway.

Reply · Share ›

Andrea · 9 years ago

Hi ... (fill with your name ;-)

I've updated my Chili script to 1.4, and have included a new recipe for MySQL (4.1), together with an example extracted from this article. I've referenced the article in the example: I hope you'll be glad. If you're not, tell me and I'll remove it :-)

Bye,
Andrea

Reply · Share ›

Xaprb · 9 years ago

Chili looks really nice, Andrea. I'm thinking about using it. (I have been thinking about using a syntax-highlighting script for a while, but don't like the one I typically see, because it uses a TEXTAREA and creates a TABLE from it). The only question is when I'll get time to actually redesign this site. Would you believe, a friend of mine who's a designer at Yahoo actually gave me a design more than a year ago and I haven't done it yet.

Reply · Share ›

Andrea · 9 years ago

For using Chili in WordPress you could use my Enzymes plugin. It's a simple filter that let's you metabolize (transclude or execute) custom fields. I wrote an article explaining how to:
http://www.mondotondo.com/aerc...

I had to add a space in your line `insert into fruits(type, variety, price) values`, because MySQL documentation says that a word followed by a parenthesis is a function, and this is now the rule that Chili applies for highlighting functions. Actually I don't like this method, and I've looked for a list of functions in the internet, but couldn't find one that was easy to add to the recipe.

Reply · Share ›

Xaprb · 9 years ago

I don't know the easiest way to get the list, but the MySQL manual certainly has all the functions in it, in section 12. It might be easiest to get the full list from the source files (I don't have the source handy to look

at right now).

If I use Chili, it won't kill me to put a space before the column list :o)

Andrea · 9 years ago

I've just made another recipe for SQL 5.1 with all the keywords and functions from the 'lex.h' source file.

I've undone the addition of the space in your line, and tested again. Now all is properly highlighted, but maybe too many things! Your 'type' column identifier is now recognized as a keyword... so I'll leave it wrapped in back quotes.

Danno · 9 years ago

Thanks so much for taking the time to portray this by such effective means - socratically. It is refreshing, and far better than the traditional way of just plopping the trophy onto the table in front of us; instead you're showing us the *truly useful* knowledge - the investigatory pathway to discovery!! Now that's SQL gravitas... I will be checking back here often, and recommending the site to my coworkers.

Xaprb · 9 years ago

Thanks! I never thought of myself as a Socrates, thinking instead that this article (written, as it was, while I figured all this out) was just bumbling about. But perhaps that's how Socrates did things, too.

fenway · 9 years ago

I've confused about the index on (type,price) -- your PK already includes (type, variety), so I had to add use index (or the equilvalent ignore index) to get it to work. Could you explain this? I know that in general, I'll have a UID field, and it won't be an issue... I think.

fenway · 9 years ago

Also, I've been having some issue on v4.1... though I don't see anything that would imply anything version-specific.

Xaprb · 9 years ago

Depending on whether you have an index that can be used for ordering, your query may use a temporary table and filesort; in that case, the variables are evaluated at a different time in the query execution plan. Use EXPLAIN to find out.

Xaprb · 9 years ago

What issue are you having? This technique is definitely playing with fire, so there could well be differences between versions.

fenway · 9 years ago

Ok, so if it goes the temp table route, it appears as though this technique doesn't "insulate" it enough from the main query itself. Which suggests that there _has_ to be a useful index or I'm stuck... correct?

As for the v4.1 issues, it simply doesn't work -- I'll have to post some specific examples, but it doesn't look like the counter gets updated at the right point in the execution plan. Once again, thanks for all your efforts -- this is a fantastic post, and blog in general; and if I can get it to work on 4.1, I'll be even more amazed.

Xaprb · 9 years ago

I'll poke around on a 4.1 server I have access to and see what I can get to happen, if I get a chance (time's short!).

Thanks for writing in and bringing this up!

fenway · 9 years ago

So it turns out that the query does work on 4.1 -- at least for ordering by (type, price ASC), with an explicit FORCE INDEX. But the equivalent query for descending doesn't work...

1

Ernest Leitch · 9 years ago

I found your site while looking for user type variables in HAVING statements because I had a problem trying to use them in a having statement.

It wasn't the silver bullet solution I was looking for, but it lead me to figure out where to put my variable evaluations so it would bring back the right values.

I'm glad you figured this out because I wouldn't have thought to hide the variables in and IS NOT NULL statement. I placed them in a LEFT JOIN and they started evaluation correctly and now it's the sweetest query I've ever written.

Nwfrg · 8 years ago

I looked around a bit and I can't find anything about the colon equals (:=) you use in your example.

What the difference between...

set @num := 0

and

set @num = 0

Xaprb · 8 years ago

See http://dev.mysql.com/doc/en/us...

Dan · 8 years ago

Excellent investigative work, thank you.

One issue that concerns me is that within the call to GREATEST(), the order of evaluation of each argument is assumed to be left to right. So that in

greatest(
@num := if(@type = type, @num 1, 1),
least(0, length(@type := type)))

The comparison @type = type is done before the new assignment, within length(), within least() of @type := type. Without this order, this would not work.

I could not find anything in the mysql reference manual that confirms that function parameters are

evaluated left to right.

Clearly this works, but are we depending on undocumented behaviour subject to change or have I missed something?

Thanks

Reply · Share ›

Xaprb · 8 years ago

Dan, yes we're depending on undocumented behavior.

Reply · Share ›

Josh Strike · 8 years ago

First of all, brilliant; I don't know how you reached this hack but it's great. Now we can do a massive recursive update in one query instead of maybe a few hundred queries. AWESOME.

The biggest problem, of course, is the ordering thing, and the fact that you can't order an update query... so it has to be really carefully constructed.

I noticed too that there were some issues, at least in my implementation, where @type would be set at the same time as @num, which could result in some weird behaviour.

Rather than putting @type:=type IS NOT NULL in a separate where clause, doing this seems to help ensure the execution order:

select *, @num
from fruits
where
(@num := if(type = @type, @num 1, IF(@type := type,1,1))) is not null
and (@num

Reply · Share ›

Josh Strike · 8 years ago

I should have said... can't order a multi-table update. Actually, I'm still banging my head against that one...without which this method won't work...Any ideas for how to force order in that case so this could work?

Reply · Share ›

Xaprb · 8 years ago

You can do things in "derived tables" and order them inside that table; then join against the table you want to update. MySQL will (probably, sometimes) put the derived table first in the join order.

To be clear:

update foo inner join (select .... [with variable magic] order by bar) as x using(somecol) set foo.y = x.z

Reply · Share ›

Josh Strike · 8 years ago

Thanks. I tried that, actually...but something really bizarre seems to happen with the ORDER BY -- at least the way I'm running it. Say I'm ordering the example by type in a subquery; The results come out in the correct order, but @num is still being incremented according to the original order. I'm thinking maybe I need to somehow have those variables eval in the order clause rather than the where...if that's somehow possible...

Reply · Share ›

Ilan Hazan · 6 years ago

Great article.
I wrote some more examples of MySQL User Defined Variables at my blog. Read
http://www.mysqldiary.com/user...

Reply · Share ›

Brooksie · 6 years ago

OK this is good, I need to do some summation and grouping - and all works just fine when run in the query browser, but when I send the query from php all variable columns come back as null...?

This is basically what I am attempting to do, although on a much larger dataset:

alter table fruit add column cost decimal(10,2);
update fruits set cost = price/2;

select type, variety, sum(price), sum(cost), @num, SUM(@profit)
from fruits
where (@num := if(type = @type, @num + 1, 1)) is not null

not null
and (@profit:= price-cost) is not null
and (@type := type) is not null
and (@num <= 5) group by type;

Ken Aston · 5 years ago

This is absolutely fantastic. I have been looking for this solution for half a year. Thank you very much.

Bill Ortell · 5 years ago

awesome, awesome, awesome... 3 days i've been trying to get something like this going - thank you for your work!

It works in under 0.002 seconds on localhost mysql dB (awesome!)

now... ;) (incoming!!! ...)

i not only follow, but implemented your example above on my data, it's perfect...

however, when at the end -
say I just wanted to do a pseudo -

" WHERE type='apple' "

before the...
" order by type, price; "

Adrian · 5 years ago

Thank you very much for this tutorial. It helped me many many times!

Jon · 5 years ago

Thanks a lot, your site has been a great aid for me so far and this post is right there at the top. However I encountered a small problem while trying to implement your solution, in every normal situation I have tries this I encounter no problem but I have one case where I run something:
SET @var = 0;
SELECT

SELECT ...
FROM t1
JOIN t1 ON (some condition)
WHERE (@var := @var = numeric_field) is not null;

This for some reason (I guess the join to the same table does it) executes the assignment twice resulting in a double increment of the var for each row, as you can guess this is not desired behavior, do you have any clue on how I might solve that?

Jon · 5 years ago

I made a mistake in my latest comment, I meant:
@var := @var + numeric_field

Michel Kogan · 5 years ago

Thanks dude, great post, it solves my problem :-)

Mike Raynham · 5 years ago

Thanks for a great article. You can use your technique if you want to reset the variables in a single query, rather than creating two statements. This might be useful for ensuring that the variables are reset each time the query is run, rather than having to remember to do two queries.

Instead of:

set @num := 0, @type := '';
select ...

You can use a UNION to do both in a single query:

select
null as type,
null as variety,
null as price,
null as num
from dual where ((@num := 0) and (@type := ''))
union
select ...

The WHERE clause creates an "Impossible WHERE", so it doesn't return any results, but it does reset the variables each time the query is run.

The alternative is to tag an extra SELECT after the FROM:

from fruits, (select @num := 0, @type := '') vars

But I think this creates a subquery, which is what you are trying to avoid in the first place.

Cabeza · 5 years ago

I see the original post appears rather old, so maybe the thread is dead, but just in case...

I found your blog when searching for an odd side-effect of using user variables in (rather simple) queries -they get very slow.

As way of example, we join a couple of tables, one of them with about a million recs the other one < 300 recs.

Here is the original version:

set @version = &#039100952'
select j.wo, t.wo from
(
select distinct wo from t1.details where version = @version and reg_id 'foo'