Working around WordPress' wpdb limitations with MySQL User Variables

Think about retrieving a data set from MySQL, where you need to add distinct, auto-incremented ID to each row retrieved. Sounds easy?

Well, the ID has to be generated on-the-fly and out of nowhere. Still easy?

One might think about using MySQL user variables to accomplish this, sure. But what if your framework does not allow injecting chained queries because of SQL injection countermeasures?

The framework I used, WordPress' wpdb API to be exact, has such a restriction, which caused my to find an alternate way.

To illustrate the problem, I start with the date retrieved, which would normally look as shown below.

```
mysql> SELECT value1, value2 FROM sample_table;
+-----------------+-----------------+
| value1          | value2          |
+-----------------+-----------------+
| sample_value_1  | sample_value_2  |
| sample_value_a  | sample_value_b  |
+-----------------+-----------------+
2 rows in set (0.00 sec)
```

Now, I need a line number for each record. When talking abut the line number, I'm strictly talking about a representation of "this is the first line", "this is the second line", etc.
This particular line number must therefore not be identical to the tupel ID, if that even would exist.
So this is what I would expect:

```
+------------+-----------------+-----------------+
| line_num   | value1          | value2          |
+------------+-----------------+-----------------+
| 1          | sample_value_1  | sample_value_2  |
| 2          | sample_value_a  | sample_value_b  |
+------------+-----------------+-----------------+
```

To generate this by the means of a query, one would usually use a MySQL user variable, that would be dynamically increased for each data row.
In SQL speech the query would then look like this:

```
SELECT @line:=0;
SELECT @line:=@line+1 AS line_num, value1, value2 FROM sample_table;
+------------+-----------------+-----------------+
| line_num   | value1          | value2          |
+------------+-----------------+-----------------+
| 1          | sample_value_1  | sample_value_2  |
| 2          | sample_value_a  | sample_value_b  |
+------------+-----------------+-----------------+
```

It's actually easy and conventient, and you get real, dynamically calculated "line numbers". But this technique requires to actually run two statements to be run in a series.
If you're using your own database routines, this may not be a problem at all. But if you rely on a given framework, you may trip into SQL injection counter measures which will simply throw away a query constructed like this:

```
$my_query = "SELECT @line:=0; SELECT @line:=@line+1 AS line_num, value1, value2 FROM sample_table;";

$my_result = $pseudo_call_to_my_db_framework->execute( $my_query );
```

This was ecactly what I was trying to do with WordPress' wpdb API. However, as I found, SQL injection filters kicked in. This is absolutely not to blame the WordPress folks, of course. The filters in place serve a good purpose and thus forced me in taking a different approach.

After some searching on the net I found that I seemed to be the only one to be using users vars with wpdb :-/
Of course, while this may not be widely used, I insisted in this approach. Suggestions in adding a loop or an extra-query where inappropriate to me, because it had to fit within the existing code base. That again required me to specifically do it in one SQL query, because I simply did not want to bloat the code by adding another loop in there.

So, the question is: Can the procedure described above be performed within ONE single query.

The answer is: Yes, but you're required to invole a sub-query to achive this.

At first, leave the initial variable assignment away, so you end up with this query:

```
SELECT @line:=@line+1 AS line_num, value1, value2 FROM sample_table;
```

This would of course not yet work, because the user variable @line wouldn't yet be defined at runtime.
Since you can't prepend the declaration, you need to embed it using a sub-query. This is needed to have the variable declaration ready at the earliest stage possible, namely while parsing the query, and not during result processing.

Let's look at the final query and the comments:

```
SELECT
    @line:=@line+1 AS line_num,              # increase the @line user variable for each row retrieved
    sample_table.value1,                     # include column 'value1' from table 'sample_table'
    sample_table.value2                      # include column 'value2' from table 'sample_table'
FROM
    (SELECT @line:=0) AS tmp_line,           # this will fire during parsing: initialize the user var by selecting it into a dummy table
    sample_table;                            # now add our actual data table as well
```

This way, you end up getting the same result as shown earlier. And yet, this query can be performed with WordPress' wpdb API and

maybe other frameworks as well.