Grouping digits in SQL

Sep 20, 2023

PostgreSQL 16 was released last week. This is the story of a feature.

One of the minor new features in PostgreSQL 16 that I am excited about is the ability to group digits in numeric literals by separating them with underscores, like

```
SELECT 1_000_000;
SELECT * FROM tbl WHERE id > 10_000_000;
```

It also works with non-integer literals:

```
SELECT * FROM tbl WHERE id < 3.1415_9265;
```

This syntax is pretty standard in contemporary programming languages. (As an exception, C and C++ use apostrophes, like `1'000'000`.)

I noticed the need for something like this a few years ago. I (and others) sometimes do simplistic benchmarks that involve generating or iterating over like a few million rows and then seeing how long that runs. So it might start like:

```
INSERT INTO tbl SELECT generate_series(1, 1000000);
```

As computers get ever faster, the number of rows you will need to get any meaningful measurements gets ever bigger. So before long you might be trying

```
INSERT INTO tbl SELECT generate_series(1, 100000000);
```

and then it gets hard to read and edit, and you might make mistakes, and the whole thing ends up wasting time and effort.

I happened to come across a proposal in the Go language, which discussed adding the same feature to that language. That discussion thankfully already worked out various details and subtleties, so I didn’t have to start the research from scratch.

So I began to think that I would like to bring this syntax to PostgreSQL.

One approach would be to write a patch and propose it to the PostgreSQL hackers list. But that would have been risky, because the hackers community is generally wary of extending core SQL syntax in ways to might conflict with the future evolution of the SQL standard. (I am myself often supportive of that wariness.)

So I planned the other approach: Get this into the SQL standard first.

At the time, I was a fairly new participant in the SQL standard working group, and such a change might have been an ambitious first project. Fortunately, another participant was just starting to work on the same issue for the GQL standard (which is managed by the same working group), so I jumped right in and we collaborated and after a few months of back and forth got this change accepted into both the SQL and GQL drafts in August 2020.

A few months later, I started to implement this in PostgreSQL, but immediately ran into a problem. To my surprise, I found that at that time, PostgreSQL allowed having numeric constants and identifiers adjacent without separating whitespace. For example, this was accepted:

```
SELECT 123abc;
```

This would parse the same as

```
SELECT 123 abc;
```

which in turn has the same meaning as

```
SELECT 123 AS abc;
```

I don’t know the history of that. But I know that the ability to omit the `AS` was added later. So maybe before that, this syntax was accepted but later failed with a parse error for other reasons, and so it was not a big deal.

What does this have to do with my project? Well, an underscore is a valid start of an identifier, and so

```
SELECT 1_000;
```

was already valid and would parse as

```
SELECT 1 _000;
```

meaning

```
SELECT 1 AS "_000";
```

So this was going to be not quite straightforward, and I parked the project for a bit.

In early 2021, the first CD ballot of ISO/IEC 9075 (SQL) finished, which is sort of a “beta” of an upcoming standard, and there were no comments submitted about “my” numeric grouping feature. That was an indication that the feature seemed pretty settled, and I started another attempt at getting this into PostgreSQL.

The first step was to fix the weird existing parsing behavior. We got consensus that this behavior was undesirable, so we just turned this into an error. We got this committed in February 2022 and it shipped with PostgreSQL 15. My idea was to give a one-major-version transition period before implementing the new feature. (I have since encountered one customer who used the old parsing behavior, but it appeared to have been a mistake.)

Then we could attack the core feature for PostgreSQL 16. The actual parser change was pretty simple, but there was more. The SQL standard also requires that the same syntax be accepted in casts from character strings to numeric types. This means we also need to accept these in the input functions of `int2`, `int4`, `int8`, and `numeric`. These functions, in particular the integer ones, are nowadays (since PostgreSQL 12) carefully tuned for performance, since they affect for example COPY FROM performance. So there was quite a bit of thought, discussion, and review needed to get this integrated in the best way. (My thanks in particular to Dean Rasheed and David Rowley for helping with all that.) We finally go this in in February 2023, and now it’s shipping in PostgreSQL 16.

(There was also a nontrivial side quest to get this feature into the SQL/JSON path language, both in PostgreSQL and the SQL standard. SQL/JSON is part of the SQL standard, but its syntax is based on the JavaScript (ECMAScript) standard, which has slightly different rules for numeric literals. Also, the PostgreSQL implementation of SQL/JSON had some corner-case bugs that were discovered during this project and were fixed in PostgreSQL 15. We got everything into PostgreSQL 16.)

Meanwhile, the new SQL standard, SQL:2023, containing “my” feature, also shipped this year, so everything is synchronized and up to date.

So that was the story of my 4-year journey that get that feature out into the world. I hope you find it useful, and maybe other SQL implementations adopt it as well.

Peter Eisentraut

Peter Eisentraut
peter@eisentraut.org

petere
petereisentraut

Peter Eisentraut's blog