PERCONA

MERGE() - A Quick Introduction

Dave Stokes
@Stoker
David.Stokes@Percona.com

Talk Proposal

MERGE() is a powerful function for processing data like transaction logs.

It is powerful as it allows you to insert, update, or delete data in one statement instead of using an application that requires numerous round trips between the server and the application to do the same tasks (which is much more resource and time-intensive).

However, this is *not* an easy function to learn from reading the manual.

This session starts with the basics and builds so that you learn how to use the power of MERGE().

20 Minutes!

Merge

Conditionally insert, update, or delete rows of a table

https://www.postgresql.org/docs/current/sql-merge.html

```
[ WITH with_query [, ...] ]
MERGE INTO [ ONLY ] target_table_name [ * ] [ [ AS ] target_alias ]
USING data_source ON join_condition
when_clause [...]

where data_source is:

{ [ ONLY ] source_table_name [ * ] | ( source_query ) } [ [ AS ] source_alias ]

and when_clause is:

{ WHEN MATCHED [ AND condition ] THEN { merge_update | merge_delete | DO NOTHING } |
  WHEN NOT MATCHED [ AND condition ] THEN { merge_insert | DO NOTHING } }

and merge_insert is:

INSERT [( column_name [, ...] )]
[ OVERRIDING { SYSTEM | USER } VALUE ]
{ VALUES ( { expression | DEFAULT } [, ...] ) | DEFAULT VALUES }

and merge_update is:

UPDATE SET { column_name = { expression | DEFAULT } |
             ( column_name [, ...] ) = ( { expression | DEFAULT } [, ...] ) } [, ...]

and merge_delete is:

DELETE
```

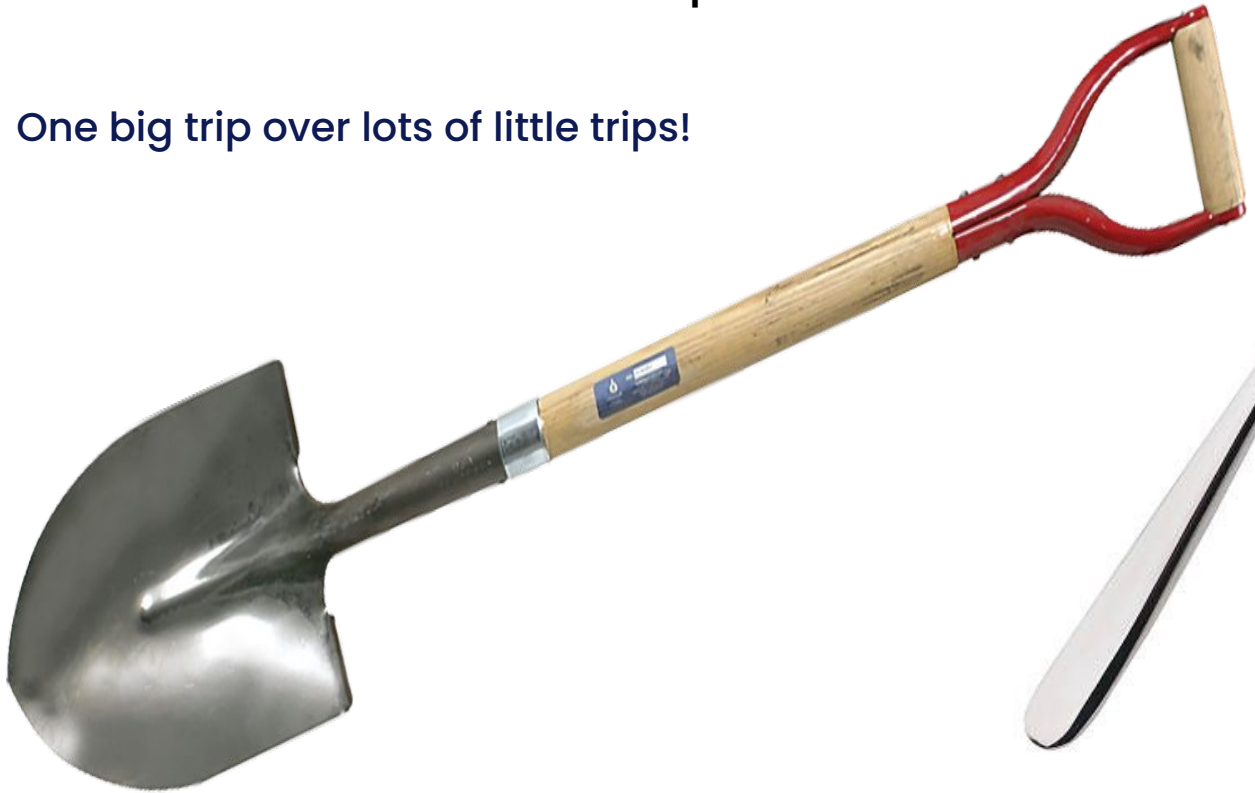
Not an UPSERT - https://www.postgresql.org/docs/current/sql-insert.html

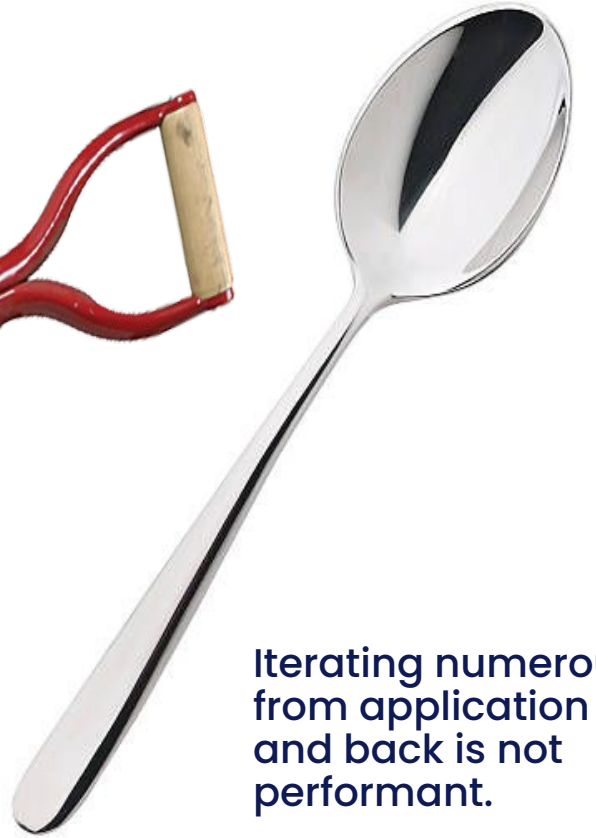
```
-- Don't update existing distributors based in a certain ZIP code
INSERT INTO distributors AS d (did, dname) VALUES (8, 'Anvil
Distribution')
    ON CONFLICT (did) DO UPDATE
    SET dname = EXCLUDED.dname || ' (formerly ' || d.dname || ')'
    WHERE d.zipcode <> '21201';

-- Name a constraint directly in the statement (uses associated
-- index to arbitrate taking the DO NOTHING action)
INSERT INTO distributors (did, dname) VALUES (9, 'Antwerp Design')
    ON CONFLICT ON CONSTRAINT distributors_pkey DO NOTHING;
```

Think Shovel Versus Spoon!

One big trip over lots of little trips!

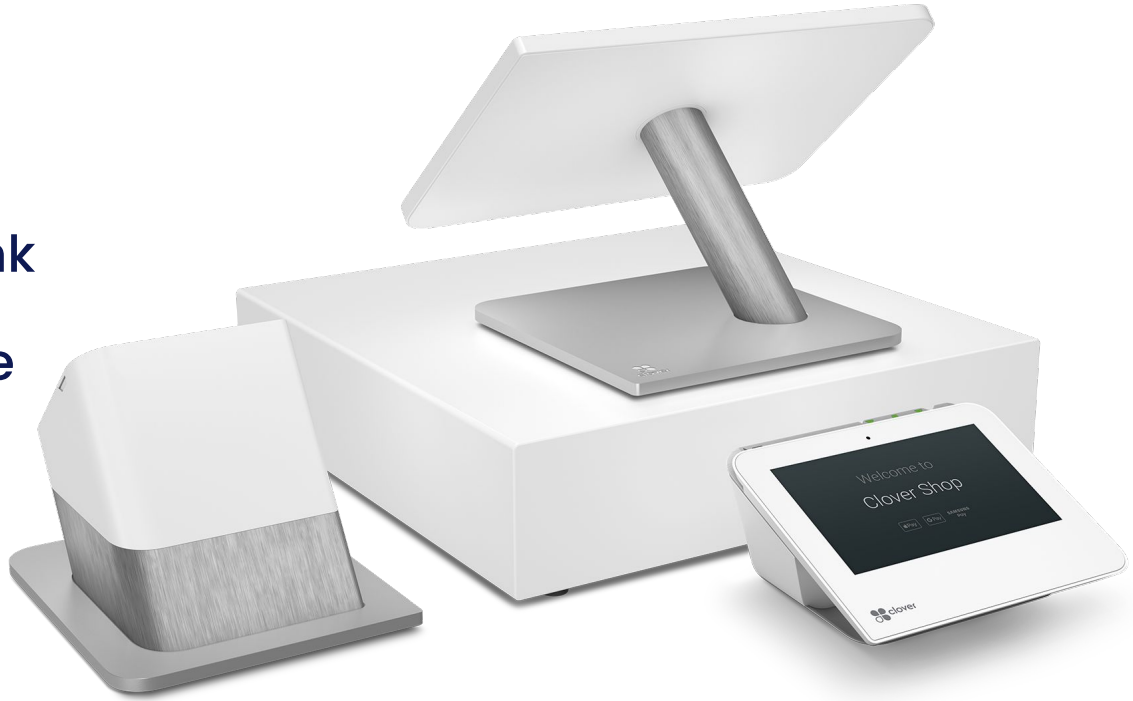




Iterating numerous times from application to server and back is not performant.

Transaction Log

For the rest of this presentation please think of ‘transaction log’ as data from a point of sale (POS) system not a databases transaction log.



PERCONA

Now that you know all!

Let us look at the big picture.

Example 1

Transaction Log

```
CREATE TABLE t (
        Id int
        x int,
        status CHAR(10)
);

Insert into t (id,x,status)
Values (1, 1, 'From log');
```

Data

```
CREATE TABLE d (
        Id int,
        x int,
        status CHAR(10),
        nbr int
);
```

In this example the tables are the same but in real-life they will not be.

Table *t* has data, table *d* does not!

Table t

```
test=# select * from t;
 id | x |    status
----+---+------------
  1 | 1 | From log
(1 row)


test=#
```

Table d

```
test=# select * from d;
 id | x | status | nbr
----+---+--------+-----
(0 rows)


test=#
```

```
MERGE into d using t on d.id = t.id
when matched then
    update set x = d.x + 1
when not matched then
   insert (id,x,status) values (t.id,t.x,t.status);
```

```
test=# MERGE into d using t on d.id = t.id
test-# when matched then
test-#     update set x = d.x + 1
test-# when not matched then
test-#    insert (id,x,status) values
(t.id,t.x,t.status);
MERGE 1
test=#
```

```
test=# select * from t;
 id | x |   status
----+---+------------
  1 | 1 | From log
(1 row)
```

```
test=# select * from d;
 id | x |   status   | nbr
----+---+------------+-----
  1 | 1 | From log   |
(1 row)
```

Merge() again!

Let’s run the same thing again

```
MERGE into d using t on d.id = t.id
when matched then
   update set x = d.x + 1
when not matched then
    insert (id,x,status) values (t.id,t.x,t.status);
```

```
test=# select * from t;
 id | x |   status
----+---+------------
  1 | 1 | From log
(1 row)
```

```
test=# select * from d;
 id | x |   status   | nbr
----+---+------------+
-----
  1 | 1 | From log   |
(1 row)
```

```
test=# MERGE into d using t on
d.id = t.id
test-# when matched then
test-#      update set x = d.x + 1
test-# when not matched then
test-#      insert (id,x,status)
values (t.id,t.x,t.status);
MERGE 1
test=# select * from d;
 id | x |   status   | nbr
----+---+------------+-----
  1 | 2 | From log   |
(1 row)
```

Deletes!

Let’s run the same thing again

Instead of updating, delete match

```
merge into d using t on d.id = t.id
when matched then delete
when not matched then insert (id,x,status) values (t.id,t.x,t.status);


test=# merge into d using t on d.id = t.id
test-# when matched then delete
test-# when not matched then insert (id,x,status) values (t.id,t.x,t.status);
MERGE 1

test=# select * from d;
 id | x | status | nbr
----+---+--------+-----
(0 rows)


test=#
```

Nothing!

Yes, nothing is an option!

test=# select * from t;

```
 id | x  |   status
----+----+------------
  1 |  1 | From log
  2 | 20 | From log
  3 | 30 | Also log
(3 rows)
```

test=# insert into d values (1,1,'in data');
INSERT 0 1

```
test=# select * from t;
 id | x  |   status
----+----+------------
  1 |  1 | From log
  2 | 20 | From log
  3 | 30 | Also log
(3 rows)

test=# insert into d values (1,1,'in data');
INSERT 0 1
test=#

test=# MERGE into d using t on t.id = d.id
test-#       when matched then
test-#     DO NOTHING
test-# when not matched then
test-# insert (id,x,status) values (t.id,t.x,t.status);
MERGE 3
test=# select * from d;
 id | x  |   status    | nbr
----+----+-------------+-----
  1 |  1 | in data     |          We already had a d.id = 1 -> do nothing!
  2 | 20 | From log    |
  3 | 30 | Also log    |
(3 rows)
```

Complicated

As complicated as you want!

New data

Clear out data

```
test=# truncate d;
TRUNCATE TABLE
test=# select * from d;
 id | x | status | nbr
----+---+--------+-----
(0 rows)
```

New transaction

```
test=# truncate t;
TRUNCATE TABLE
test=# insert into t values (1,1,'original');
INSERT 0 1
test=#
```

New merge

```
merge into d      #Did not change
using t
on t.id = d.id

when matched AND d.x > 2 THEN
UPDATE SET x = d.x + t.x, status='updated+'

when matched and d.x = 1 THEN
UPDATE SET status = 'updated', x = 3

when not matched then
insert (id,x,status) values (t.id,t.x,t.status);
```

First use

Merge

```
merge into d
using t
on t.id = d.id
when matched AND d.x > 2 THEN
UPDATE SET x = d.x + t.x, status='updated+'
when matched and d.x = 1 THEN
UPDATE SET status = 'updated', x = 3
when not matched then
insert (id,x,status) values (t.id,t.x,t.status);
```

Data

```
test=# select * from d;
 id | x |   status   | nbr
----+---+------------+-----
  1 | 1 | original   |
(1 row)
```

Second use

Merge

```
merge into d
using t
on t.id = d.id
when matched AND d.x > 2 THEN
UPDATE SET x = d.x + t.x, status='updated+'
when matched and d.x = 1 THEN
UPDATE SET status = 'updated', x = 3
when not matched then
insert (id,x,status) values (t.id,t.x,t.status);
```

Data

```
test=# select * from d;
 id | x |   status   | nbr
----+---+------------+-----
  1 | 3 | updated    |
(1 row)
```

Third use

Merge

```
merge into d
using t
on t.id = d.id
when matched AND d.x > 2 THEN
UPDATE SET x = d.x + t.x,
status='updated+'
when matched and d.x = 1 THEN
UPDATE SET status = 'updated', x = 3
when not matched then
insert (id,x,status) values (t.id,t.x,t.status);
```

Data

```
test=# select * from d;
 id | x |   status   | nbr
----+---+------------+-----
  1 | 4 | updated+   |
(1 row)
```

Those are the basics

Well, at least the basic basics, at basic level

Triggers

If you love to have your ‘business logic’ in the database then please read carefully the manual page on when/how triggers are fired.

Yup, this is like juggling while bouncing up and down on a unicycle while crossing a tightrope.

PostgreSQL 17

MERGE() gets and update!!

Two New Features

RETURNING

The optional RETURNING clause causes MERGE to compute and return value(s) based on each row inserted, updated, or deleted.

Any expression using the source or target table's columns, or the merge_action() function can be computed.

When an INSERT or UPDATE action is performed, the new values of the target table's columns are used.

When a DELETE is performed, the old values of the target table's columns are used. The syntax of the RETURNING list is identical to that of the output list of SELECT.

Two New Features

MERGE_ACTION()

Returns the merge action command executed for the current row.

This will be 'INSERT', 'UPDATE', or 'DELETE'.

```
MERGE INTO products p
  USING stock s ON p.product_id = s.product_id
  WHEN MATCHED AND s.quantity > 0 THEN
    UPDATE SET in_stock = true, quantity =
s.quantity
  WHEN MATCHED THEN
    UPDATE SET in_stock = false, quantity = 0
  WHEN NOT MATCHED THEN
    INSERT (product_id, in_stock, quantity)
      VALUES (s.product_id, true, s.quantity)
  RETURNING merge_action(), p.*;

 merge_action | product_id | in_stock | quantity
--------------+------------+----------+----------
 UPDATE       |       1001 | t        |       50
 UPDATE       |       1002 | f        |        0
 INSERT       |       1003 | t        |       10
```

https://www.postgresql.org/docs/devel/functions-merge-support.html#MERGE-ACTION

Some suggested reading

https://www.postgresql.org/docs/current/sql-merge.html

https://www.percona.com/blog/using-merge-to-make-your-postgresql-more-powerful/

https://www.postgresql.fastware.com/blog/the-postgresql-merge-command-a-useful-tool-to-make-your-code-more-efficient

Innovate freely with highly available and reliable production PostgreSQL

Try Percona software:

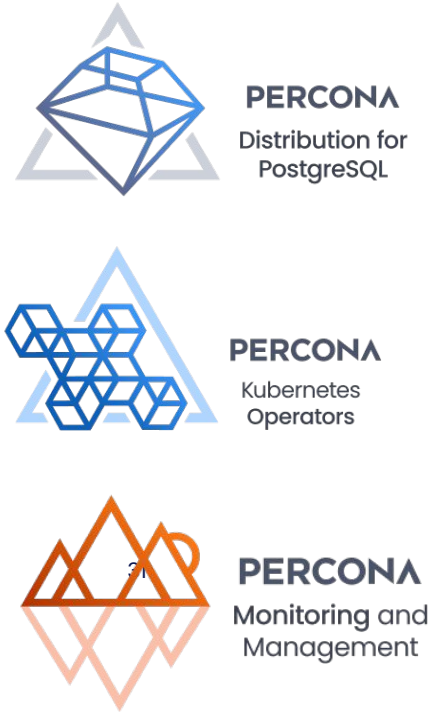
- ➔ Percona Distribution for Postgres
- ➔ Percona Operator for PostgreSQL
- ➔ Percona Monitoring and Management (PMM)

We have a TDE solution looking for testers!

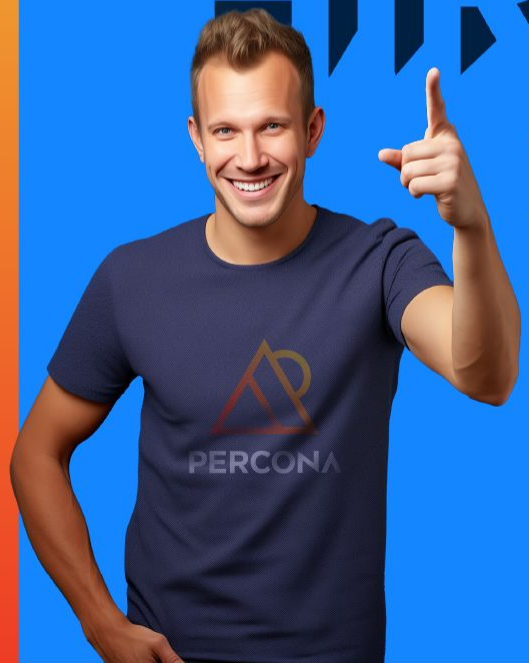
- ➔ **github.com/Percona-Lab/postgresql-tde**

Ask questions and leave your feedback:

- ➔ percona.community
- ➔ forums.percona.com
- ➔ github.com/percona



WE ARE HIRING



We are **hiring**.
Check our **openings**.

PERCONA

Current openings include:

- Senior Software Engineer (PostgreSQL)
- Support Engineer (PostgreSQL)
- PostgreSQL Evangelist
- ...and more!

Thank You!

David.Stokes@Percona.com
@Stoker
Speakerdeck.com/Stoker