MERGE: Built to Remove Barriers

Shruthi K C

23-Feb-2023 @PGConf India

EDB

About Me

My name is Shruthi K C

Working at EnterpriseDB as a Database Developer

Email: shruthi.kc@enterprisedb.com

Agenda

- Why MERGE ?
- Introduction to MERGE SQL command
- MERGE syntax
- How MERGE works ?
- Things to keep in mind while using MERGE
- Applications of MERGE
- Current Limitations
- Wish list for Future
- Q&A

Why MERGE ?

- Lack of a MERGE statement in Postgres forced developers to create custom workarounds

- Cost of migration was highly expensive from both a time and resource perspective.

- Absence of MERGE in Postgres became a **barrier** for Customers from migration to Postgres.

- **A long time in the making, the much-requested MERGE feature has finally made its way to PostgreSQL.**

- **POSTGRES has full support for customer migrating to postgres** and the introduction of **MERGE feature is likely to accelerate the Postgres adoption** in the enterprise market.

Introduction to MERGE SQL Command

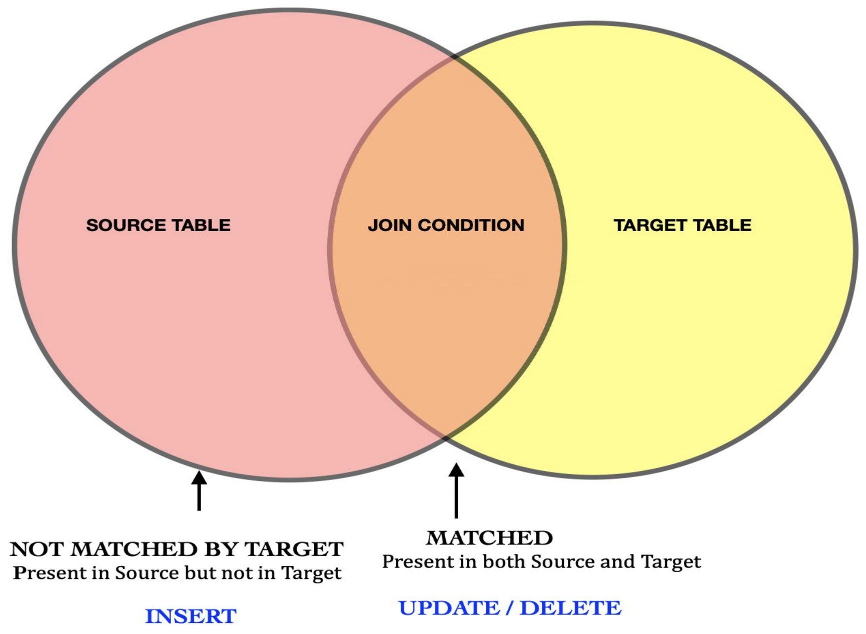
- MERGE provides a **single SQL statement** that can conditionally **INSERT, UPDATE and DELETE rows,** eliminating the need to write separate logic for each.

- MERGE actions have the same effect as regular **UPDATE, INSERT,** or **DELETE** commands of the same names.

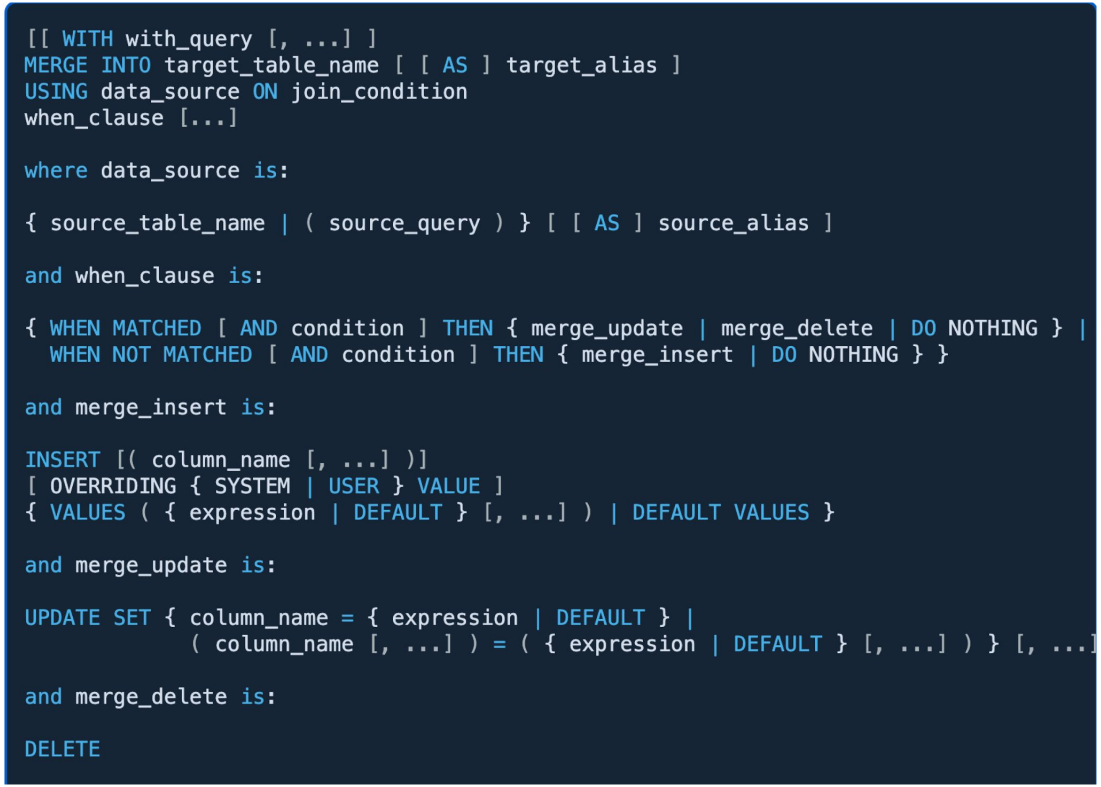
- MERGE simplifies SQL scripts for database developers and administrators and the production code can be more easily maintained.

- MERGE statement boost the **performance** by reading and processing data in a single query and avoids multiple I/O operations from the disk.

Deep dive to MERGE SQL command



MERGE Syntax



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

where data_source is:

{ 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
```

Compatibility

- MERGE sql command conforms to the SQL standard.
- The **WITH** clause and **DO NOTHING** action are extensions to the SQL standard.

UPDATE using MERGE

SOURCE TABLE		
id	**name**	**price**
1	Mango	10
2	Orange	20
3	Apple	35

TARGET TABLE		
id	**name**	**price**
1	Mango	25
2	Orange	15
6	Avocado	60

```
MERGE INTO target t
USING source s
ON (s.id = t.id)
WHEN MATCHED THEN
    UPDATE SET price = s.price;
```

MERGE RESULT →

TARGET TABLE		
id	**name**	**price**
1	**Mango**	**10**
2	**Orange**	**20**
6	Avocado	60

Conditional UPDATE using MERGE

SOURCE TABLE		
id	**name**	**price**
1	Mango	10
2	Orange	20
3	Apple	35

TARGET TABLE		
id	**name**	**price**
1	Mango	25
2	Orange	15
6	Avocado	60

```
MERGE INTO target t
USING source s
ON (s.id = t.id)
WHEN MATCHED AND s.name != ‘Orange’ THEN
        UPDATE SET price = s.price;
```

MERGE RESULT →

TARGET TABLE		
id	**name**	**price**
1	**Mango**	**10**
2	Orange	15
6	Avocado	60

DELETE using MERGE

SOURCE TABLE		
id	**name**	**price**
1	Mango	10
2	Orange	20
3	Apple	35

TARGET TABLE		
id	**name**	**price**
1	**Mango**	**25**
2	**Orange**	**15**
6	Avocado	60

```
MERGE INTO target t
USING source s
ON (s.id = t.id)
WHEN MATCHED THEN
      DELETE;
```

MERGE RESULT →

TARGET TABLE		
id	**name**	**price**
6	Avocado	60

Conditional DELETE using MERGE

SOURCE TABLE

id	name	price
1	Mango	10
2	Orange	20
3	Apple	35

TARGET TABLE

id	name	price
1	**Mango**	**25**
2	Orange	15
6	Avocado	60

```
MERGE INTO target t
USING source s
ON (s.id = t.id)
WHEN MATCHED AND t.price > 20 THEN
      DELETE;
```

MERGE RESULT

TARGET TABLE

id	name	price
2	Orange	15
6	Avocado	60

INSERT USING MERGE

SOURCE TABLE		
id	**name**	**price**
1	Mango	10
2	Orange	20
3	**Apple**	**35**

```
MERGE INTO target t
USING source s
ON (s.id = t.id)
WHEN NOT MATCHED THEN
        INSERT (id, name, price)
        VALUES (s.id, s.name, s.price);
```

TARGET TABLE		
id	**name**	**price**
1	Mango	25
2	Orange	15

MERGE RESULT →

TARGET TABLE		
id	**name**	**price**
1	Mango	10
2	Orange	20
3	**Apple**	**35**

Conditional INSERT USING MERGE

SOURCE TABLE		
id	**name**	**price**
1	Mango	10
2	Orange	20
3	**Apple**	**35**
4	**Grapes**	**20**
5	**Papaya**	**15**

TARGET TABLE		
id	**name**	**price**
1	Mango	25
2	Orange	15

```
MERGE INTO target t
USING source s
ON (s.id = t.id)
WHEN NOT MATCHED AND s.price < 30 THEN
        INSERT (id, name, price)
        VALUES (s.id, s.name, s.price);
```

MERGE RESULT →

TARGET TABLE		
id	**name**	**price**
1	Mango	10
2	Orange	20
4	**Grapes**	**20**
5	**Papaya**	**15**

Merging into a Table: Example

SOURCE TABLE

id	name	price
1	Mango	11
2	Orange	12
3	Apple	13
4	Banana	14
5	Grapes	15

TARGET TABLE

id	name	price
1	Mango	10
2	Orange	6
3	Apple	7

```
MERGE INTO target t
USING source s
ON (s.id = t.id)
WHEN MATCHED AND t.price < 10 THEN
        UPDATE SET price = s.price
WHEN MATCHED THEN
        DELETE
WHEN NOT MATCHED THEN
        INSERT (id, name, price)
        VALUES (s.id, s.name, s.price);
```

🟧	WHEN MATCHED UPDATE
🟥	WHEN MATCHED DELETE
🟩	WHEN NOT MATCHED INSERT

MERGE RESULT →

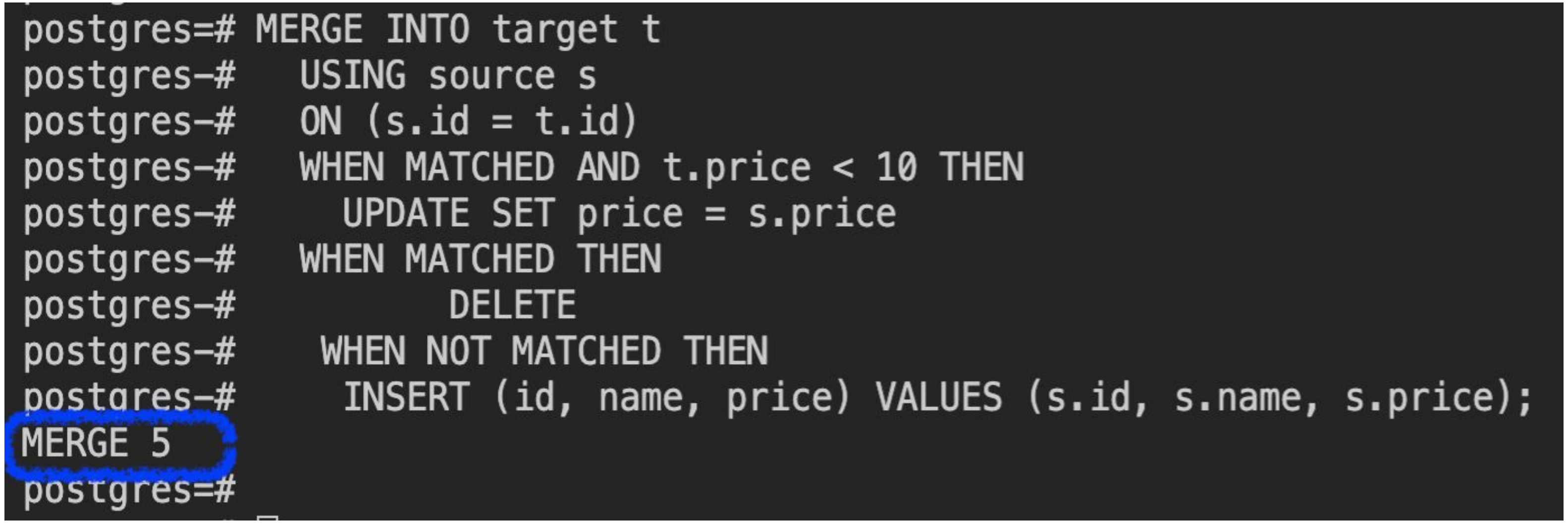
TARGET TABLE

id	name	price
2	Orange	12
3	Apple	13
4	Banana	14
5	Grapes	15

EDB

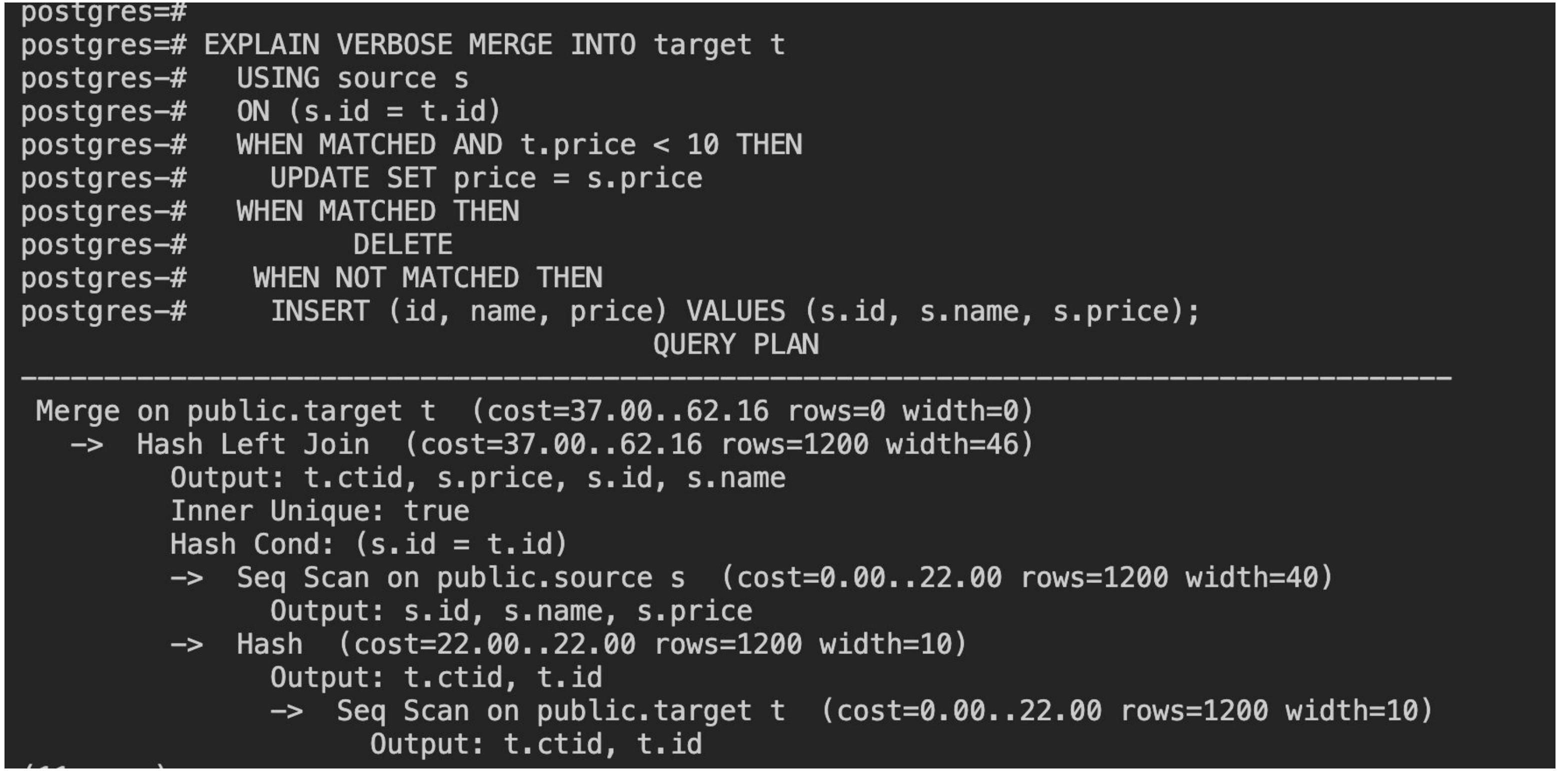
The MERGE Output

On successful completion, a MERGE command returns **MERGE total_count.** The total_count is the total number of rows changed (whether inserted, updated, or deleted).



```
postgres=# MERGE INTO target t
postgres-#   USING source s
postgres-#   ON (s.id = t.id)
postgres-#   WHEN MATCHED AND t.price < 10 THEN
postgres-#     UPDATE SET price = s.price
postgres-#   WHEN MATCHED THEN
postgres-#          DELETE
postgres-#    WHEN NOT MATCHED THEN
postgres-#     INSERT (id, name, price) VALUES (s.id, s.name, s.price);
MERGE 5
postgres=#
```

Query Plan for MERGE



```
postgres=#
postgres=# EXPLAIN VERBOSE MERGE INTO target t
postgres-#   USING source s
postgres-#   ON (s.id = t.id)
postgres-#   WHEN MATCHED AND t.price < 10 THEN
postgres-#     UPDATE SET price = s.price
postgres-#   WHEN MATCHED THEN
postgres-#          DELETE
postgres-#    WHEN NOT MATCHED THEN
postgres-#     INSERT (id, name, price) VALUES (s.id, s.name, s.price);
                                       QUERY PLAN
--------------------------------------------------------------------------------------
 Merge on public.target t  (cost=37.00..62.16 rows=0 width=0)
   ->  Hash Left Join  (cost=37.00..62.16 rows=1200 width=46)
         Output: t.ctid, s.price, s.id, s.name
         Inner Unique: true
         Hash Cond: (s.id = t.id)
         ->  Seq Scan on public.source s  (cost=0.00..22.00 rows=1200 width=40)
               Output: s.id, s.name, s.price
         ->  Hash  (cost=22.00..22.00 rows=1200 width=10)
               Output: t.ctid, t.id
               ->  Seq Scan on public.target t  (cost=0.00..22.00 rows=1200 width=10)
                     Output: t.ctid, t.id
```

Rows affected by MERGE

- MERGE performs actions only on the original rows at the time of JOIN.

- New records inserted in MERGE cannot be updated/deleted in the same merge statement.

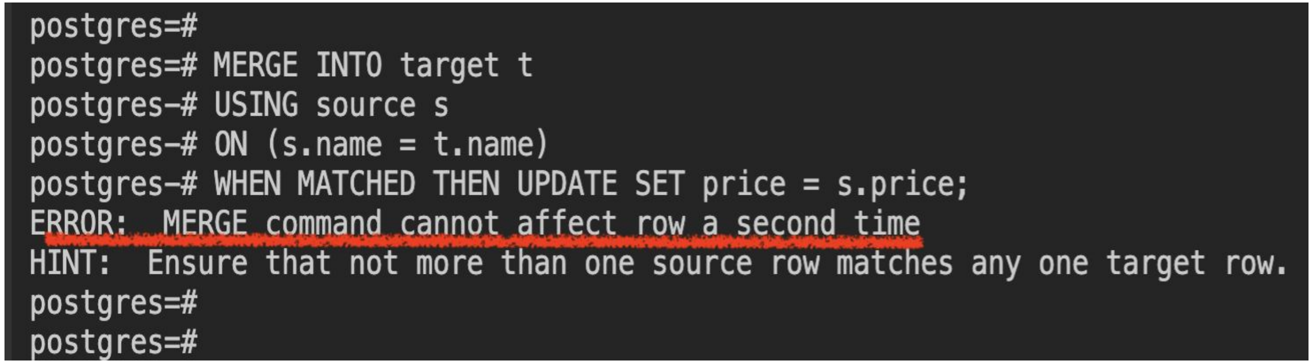
- Records updated in the MERGE cannot be deleted in the same merge statement.

Update/Delete Each Row Once

SOURCE TABLE		
id	**name**	**price**
1	Mango	10
2	Mango	20
3	Apple	35

TARGET TABLE		
id	**name**	**price**
1	Mango	25
6	Avocado	60

MERGE INTO target t
USING source s
ON **(s.name = t.name)**
WHEN MATCHED THEN
UPDATE SET price = s.price;



```
postgres=#
postgres=# MERGE INTO target t
postgres-# USING source s
postgres-# ON (s.name = t.name)
postgres-# WHEN MATCHED THEN UPDATE SET price = s.price;
ERROR:  MERGE command cannot affect row a second time
HINT:  Ensure that not more than one source row matches any one target row.
postgres=#
postgres=#
```

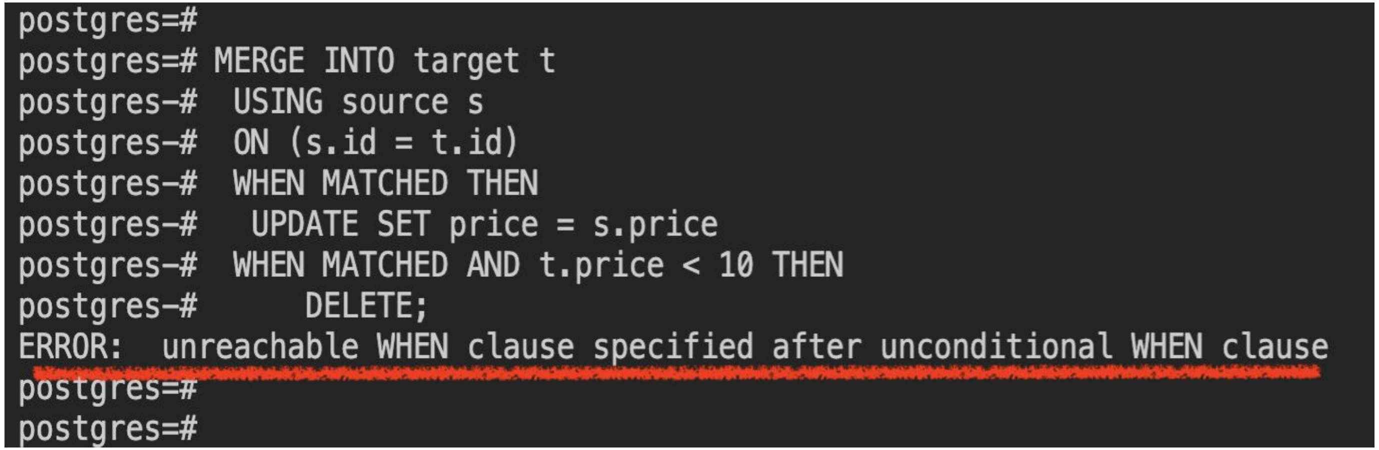
A word of Caution

- A primary key, unique key, or unique index isn’t mandatory for a MERGE statement. However, ensuring **unique key constraints on join columns** can avoid target row matching more than one source row.

- Creating proper indexes on both tables and **join only the required columns** can avoid running into performance issues while synchronizing the tables.

Avoid Unreachable WHEN CLAUSE

- If a WHEN clause omits an AND sub-clause, it becomes the final reachable clause of that kind (MATCHED or NOT MATCHED) and the following WHEN clause becomes unreachable.
- **If two clauses are specified, the first clause must be accompanied by an AND <search_condition> clause**.



```
postgres=#
postgres=# MERGE INTO target t
postgres-#  USING source s
postgres-#  ON (s.id = t.id)
postgres-#  WHEN MATCHED THEN
postgres-#   UPDATE SET price = s.price
postgres-#  WHEN MATCHED AND t.price < 10 THEN
postgres-#      DELETE;
ERROR:  unreachable WHEN clause specified after unconditional WHEN clause
postgres=#
postgres=#
```

MERGE Privilege

- There is no separate MERGE privilege.

- SELECT privilege on the source table.

- UPDATE privilege on the target table for update action, the INSERT privilege for insert action and/or the DELETE privilege if you wish to delete.

MERGE with Triggers

- **BEFORE STATEMENT** triggers are performed for all actions specified, whether or not their WHEN clauses match.

- **BEFORE ROW** triggers are performed for the action’s event type if their WHEN clauses match.

- **AFTER ROW** triggers are performed for the action’s event type after the actions are performed.

- **AFTER STATEMENT** triggers are performed for all actions specified, whether or not they actually occur.

Concurrency and Isolation

- If the row is concurrently updated/deleted such that the join condition fails, then MERGE will evaluate the condition's NOT MATCHED actions.

- The conditions for each action are re-evaluated on the updated version of the row, starting from the first action.

- If MERGE attempts an INSERT and a unique index is present and a duplicate row is concurrently inserted, then a uniqueness violation error is raised.

Applications of MERGE SQL Statement

- An example of OLTP case is a table that isn't updated directly by your application and instead, you get a delta of changes periodically from an external system.

- In data warehouse, MERGE can be used to maintain Slowly Changing Dimensions (SCD).

Current Limitations

- MERGE is not supported if the target table is a view or foreign table.
- RETURNING clause is not allowed in MERGE.
- MERGE is not supported if the target table has any rules defined on it.

Wishlist for Future

Wish to see a **WHEN NOT MATCHED BY SOURCE** clause in Postgres.

Q & A

Thank You

EDB™

POWER TO POSTGRES

We are hiring. Be part of our team!

Scan Now to Apply!

careers@enterprisedb.com

@EDBPostgres /company/EDBPostgres www.enterprisedb.com