SQL Challenge: Find Diagonally Symmetric Pairs

Problem Statement

You are given a table `Functions` with two columns `x` and `y`, each representing a function mapping such as `f(x) = y`.

Find all **diagonally symmetric pairs** in the table. A diagonally symmetric pair means:

- There exists a row `(x, y)`
- And another row `(y, x)` also exists in the table

Return each such pair **only once** in ascending order of `x`, avoiding duplicates like both `(1, 2)` and `(2, 1)`.

Input Table: `Functions`

x	y
1	2
2	1
4	3
5	6

Expected Output

x	y
1	2
2	1

Only `(1, 2)` and `(2, 1)` form a symmetric pair, so they are returned.

MySQL Solution

```sql
SELECT A.x, A.y
FROM FUNCTIONS A
JOIN FUNCTIONS B
    ON A.x = B.y AND A.y = B.x
GROUP BY A.x, A.y
HAVING COUNT(*) > 1 OR A.x < A.y
ORDER BY A.x;
```

Understanding the Problem

We need to:

1. Find pairs of rows where (x, y) and (y, x) both exist in the table
2. Return each symmetric pair only once (not both (1, 2) and (2, 1))
3. Order the results by x in ascending order

Breaking Down the Solution

Let's analyze the SQL query that solves this problem:

```
SELECT A.x, A.y
FROM FUNCTIONS A
JOIN FUNCTIONS B
    ON A.x = B.y AND A.y = B.x
GROUP BY A.x, A.y
HAVING COUNT(*) > 1 OR A.x < A.y
ORDER BY A.x;
```

Step 1: Self-Join to Find Symmetric Pairs

The first key part of this solution is a self-join of the `FUNCTIONS` table:

```
FROM FUNCTIONS A
JOIN FUNCTIONS B
    ON A.x = B.y AND A.y = B.x
```

Here, we're joining the table with itself using two aliases (A and B). The join condition is designed to match rows where:

- A's x matches B's y
- A's y matches B's x

For our sample data, this join would produce:

A.x	A.y	B.x	B.y
1	2	2	1
2	1	1	2

Notice that:

- Row (1, 2) from A matches with row (2, 1) from B
- Row (2, 1) from A matches with row (1, 2) from B
- Rows (4, 3) and (5, 6) don't have matching symmetric pairs

Step 2: Eliminating Duplicates

Now we need to ensure we only keep one of each symmetric pair. The solution uses this clever combination:

```
GROUP BY A.x, A.y
HAVING COUNT(*) > 1 OR A.x < A.y
```

Let's break it down:

1. `GROUP BY A.x, A.y` - Groups the result set by x and y values from table A

2. `HAVING COUNT(*) > 1 OR A.x < A.y` - This condition is key to understanding the solution:

   - `COUNT(*) > 1`: This would catch cases where x = y (diagonal elements). If there's a row (5, 5), it would match with itself in the join, and we'd want to include it.

   - `A.x < A.y`: This is the clever part! For each symmetric pair like (1, 2) and (2, 1), this condition will only be true for one of them (the one where x < y).

For our joined result:

- For (1, 2): 1 < 2 is TRUE
- For (2, 1): 2 < 1 is FALSE

So the HAVING clause will only keep (1, 2) and filter out (2, 1).

Wait, but our expected output shows both (1, 2) and (2, 1)? Let's double-check the data and solution...

Actually, for our particular sample data, the solution returns:

x	y
1	2
2	1

This is because:

- Both (1, 2) and (2, 1) form symmetric pairs with each other
- (1, 2) meets A.x < A.y condition
- (2, 1) doesn't meet A.x < A.y, but it meets the COUNT(*) > 1 condition because there's exactly one match in the join

Step 3: Understanding a Common Confusion Point

There's a potential confusion here that I should clarify. When we GROUP BY A.x and A.y:

- For the pair (1, 2), there's one matching row (2, 1) from table B
- For the pair (2, 1), there's one matching row (1, 2) from table B

But the `COUNT(*) > 1` in our HAVING clause isn't counting these matches. It's counting the number of rows that get grouped together when we GROUP BY A.x, A.y, which would only be > 1 if we had duplicate rows in the original table.

So actually, the correct interpretation of the solution is:

```sql
SELECT A.x, A.y
FROM FUNCTIONS A
JOIN FUNCTIONS B
    ON A.x = B.y AND A.y = B.x
WHERE A.x <= A.y  -- Only keep one of each symmetric pair
ORDER BY A.x;
```

Or, for the original query with the GROUP BY:

```
HAVING COUNT(*) > 1 OR A.x < A.y
```

This is checking:

1. COUNT(*) > 1: Are there duplicate (x, y) pairs in the original table?
2. OR A.x < A.y: If not, is this the smaller x value in the symmetric pair?

For our simple example, (1, 2) and (2, 1) appear exactly once in the FUNCTIONS table, so COUNT(*) = 1 for each of them in the GROUP BY. But (1, 2) satisfies A.x < A.y, so it's included.

The Correct Result

Given our join and filtering conditions, the query returns:

x	y
1	2

Because (1, 2) is the only pair that satisfies A.x < A.y, and there are no duplicate rows where COUNT(*) > 1.

But wait - the expected output shows both (1, 2) and (2, 1). Let me review the query one more time...

The issue is with the `GROUP BY` and `HAVING` combination. This approach assumes that for symmetric pairs, we're selecting the pair where x < y. But the expected output shows both pairs.

The correct query if we want both (1, 2) and (2, 1) in the output should be something like:

```sql
SELECT A.x, A.y
FROM FUNCTIONS A
JOIN FUNCTIONS B
    ON A.x = B.y AND A.y = B.x
ORDER BY A.x;
```

This would return both symmetric pairs. The original query with the HAVING clause is trying to do additional filtering that doesn't match the expected output.

In real-world applications, you'd typically choose whether to return one or both elements of each symmetric pair based on your specific requirements.