SQL Challenge: Customers Who Never Placed an Order

Problem Statement

You are given two tables:

- `Customers` — contains customer information.
- `Orders` — contains customer orders, each referencing a customer.

Your task is to find all **customers who have never placed an order**.

Input Tables

`Customers`

id	name
1	Alice
2	Bob
3	Charlie

`Orders`

id	customerId
1	1
2	1
3	2

Expected Output

Customers
Charlie

SQL Solution

```sql
SELECT
    cust.name AS Customers
FROM Customers cust
LEFT JOIN Orders ord
    ON cust.id = ord.customerId
WHERE ord.customerId IS NULL;
```

Understanding the Problem

We need to:

1. Find all customers who have never placed an order
2. Return just their names in a column called "Customers"

Breaking Down the Solution

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

```sql
SELECT
    cust.name AS Customers
FROM Customers cust
LEFT JOIN Orders ord
    ON cust.id = ord.customerId
WHERE ord.customerId IS NULL;
```

Step 1: LEFT JOIN to Find Missing Relationships

The key technique here is using a LEFT JOIN between the Customers and Orders tables:

```sql
FROM Customers cust
LEFT JOIN Orders ord
    ON cust.id = ord.customerId
```

A LEFT JOIN includes all rows from the left table (Customers) and matching rows from the right table (Orders). If there's no match in the right table, NULL values are returned for those columns.

Let's see what this join produces with our sample data:

cust.id	cust.name	ord.id	ord.customerId
1	Alice	1	1
1	Alice	2	1
2	Bob	3	2
3	Charlie	NULL	NULL

Notice that:

- Alice (id=1) has two orders (id=1 and id=2)
- Bob (id=2) has one order (id=3)
- Charlie (id=3) has no orders, so the joined columns are NULL

Step 2: Filtering for NULL Customer IDs

After the LEFT JOIN, we identify customers with no orders by checking for NULL values in the Orders table's customer ID column:

```sql
WHERE ord.customerId IS NULL;
```

This filters out any rows where there was a match in the Orders table, leaving only customers who have never placed an order.

In our sample data, only Charlie has NULL for ord.customerId, so only Charlie will be included in the final result.

Step 3: Selecting the Desired Output

Finally, we select just the customer name and alias it as "Customers" to match the expected output format:

```
SELECT
    cust.name AS Customers
```

Step 4: Final Result

The final query returns:

Customers
Charlie

Why This Works (Detailed Explanation)

This solution leverages three key SQL concepts:

1. **LEFT JOIN**: This type of join includes all records from the left table (Customers) regardless of whether there are matching records in the right table (Orders). This allows us to see all customers, even those with no orders.
2. **NULL Filtering**: When a customer has no orders, the columns from the Orders table will contain NULL values after the LEFT JOIN. By filtering for `ord.customerId IS NULL`, we specifically select only those customers who have no corresponding entries in the Orders table.
3. **Column Aliasing**: We use `AS Customers` to rename the output column to match the expected format.

Alternative Approaches

There are several other ways to solve this problem:

Approach 2: Using NOT EXISTS

```
SELECT name AS Customers
FROM Customers c
WHERE NOT EXISTS (
    SELECT 1
    FROM Orders o
    WHERE o.customerId = c.id
);
```

This approach uses a subquery with NOT EXISTS to find customers with no matching orders. It checks if any orders exist for each customer and selects only those where none do.

Approach 3: Using NOT IN

```sql
SELECT name AS Customers
FROM Customers
WHERE id NOT IN (
    SELECT customerId
    FROM Orders
    WHERE customerId IS NOT NULL
);
```

This approach uses NOT IN to exclude customers whose IDs appear in the Orders table. The `WHERE customerId IS NOT NULL` is important when using NOT IN to avoid issues with NULL values.

Approach 4: Using Except/Minus

In some SQL dialects, you could use set operations:

```sql
SELECT name AS Customers
FROM Customers
WHERE id IN (
    SELECT id FROM Customers
    EXCEPT
    SELECT customerId FROM Orders
);
```

This finds the set difference between all customer IDs and customer IDs that have placed orders.

Which Approach Is Best?

The LEFT JOIN approach (the original solution) is typically the most efficient and readable, especially for beginners. It clearly shows what we're trying to do: "show me all customers and their orders, then filter out those who have any orders."

For very large datasets, you might want to benchmark different approaches, as performance can vary depending on indexes and the specific database system you're using.