SQL Pivot Challenge: Students by Continent

Problem Statement

You are given a table called `Student`, where each student has a `name` and belongs to a `continent`.

Your task is to **pivot** this data so that each **continent becomes a column**, and student names from each continent are displayed row-wise in **alphabetical order**. If there are fewer students in a continent, the corresponding cell should be `null`.

Input Table: `Student`

name	continent
Jane	America
Pascal	Europe
Xi	Asia
Jack	America

Expected Output

America	Asia	Europe
Jack	Xi	Pascal
Jane	null	null

T-SQL Solution

```
SELECT
    MAX(CASE WHEN continent = 'America' THEN name END) AS America,
    MAX(CASE WHEN continent = 'Asia' THEN name END) AS Asia,
    MAX(CASE WHEN continent = 'Europe' THEN name END) AS Europe
FROM (
    SELECT *,
           ROW_NUMBER() OVER (PARTITION BY continent ORDER BY name) AS rn
    FROM Student
```

Solving the SQL Pivot Challenge: Students by Continent

I'll explain this step by step so that beginners can understand how to transform rows into columns (pivot) in SQL.

Understanding the Problem

We need to transform data from this format:

- Rows containing student name and continent
- Each student is one row

Into this format:

- Continents become column headers
- Student names appear under their respective continents
- Names should be alphabetically sorted within each continent
- If a continent has fewer students, we use NULL values to fill the gaps

Breaking Down the Solution

Let's analyze this in simple, logical steps:

Step 1: Sort and Number the Students

First, we need to sort students alphabetically within each continent. We use the ROW_NUMBER() function for this:

```
SELECT *,
       ROW_NUMBER() OVER (PARTITION BY continent ORDER BY name) AS rn
FROM Student
```

This gives us:

name	continent	rn
Jack	America	1
Jane	America	2
Xi	Asia	1
Pascal	Europe	1

The `PARTITION BY continent` creates separate numbering for each continent, and `ORDER BY name` ensures alphabetical order.

Step 2: Pivot the Data

Now we need to transform this into columns. The incomplete query you provided uses the CASE statement for pivoting:

```
SELECT
    MAX(CASE WHEN continent = 'America' THEN name END) AS America,
    MAX(CASE WHEN continent = 'Asia' THEN name END) AS Asia,
    MAX(CASE WHEN continent = 'Europe' THEN name END) AS Europe
FROM (
    SELECT *,
           ROW_NUMBER() OVER (PARTITION BY continent ORDER BY name) AS rn
    FROM Student
) AS numbered_students
```

```
GROUP BY rn
ORDER BY rn
```

Let's understand what this does:

1. For each row number (rn), we look at all continents
2. We pick the name from America, Asia, and Europe that has that row number
3. We use MAX() because aggregate functions are required when using GROUP BY
4. We GROUP BY rn to create one output row for each row number
5. ORDER BY rn ensures rows appear in the correct sequence

Step 3: The Final Result

This gives us our expected output:

America	Asia	Europe
Jack	Xi	Pascal
Jane	null	null

For the first row (rn=1), we have "Jack" from America, "Xi" from Asia, and "Pascal" from Europe. For the second row (rn=2), we only have "Jane" from America, while the other continents are NULL because they don't have a second student.

Why This Works (Detailed Explanation)

1. **ROW_NUMBER()**: This function assigns sequential numbers to rows within each partition (continent), sorting by name.

2. **Subquery**: We need the row numbers first, which is why we use a subquery to calculate them.

3. **CASE Statement**: For each continent column:

   - `CASE WHEN continent = 'America' THEN name END` returns the name when the continent is America, otherwise NULL
   - Similar expressions create columns for Asia and Europe

4. **MAX() Function**: Since we're grouping by row number, we need an aggregate function. MAX() takes the non-NULL value when there is one.

5. **GROUP BY rn**: This creates one output row for each row number, combining students with the same row number across different continents.

This technique is called "conditional aggregation" and is one of several ways to pivot data in SQL.

Potential Improvements

For a more dynamic solution that doesn't require hardcoding continent names, you would need to use dynamic SQL, but that's a more advanced topic beyond the scope of this explanation.