🧠 SQL Problem: People Whose Best Friends Got Better Salaries (MySQL)

📝 Problem Statement

You are given three tables:

- **Students(id, name)** — A list of students.
- **Friends(id, friend_id)** — Each student has a best friend (one-to-one mapping).
- **Packages(id, salary)** — Each student's salary package.

Goal: Find the names of all students whose best friend got a higher salary than them.

📊 Example

Input

Students | id | name | |----|------| | 1 | A | | 2 | B | | 3 | C | | 4 | D |

Friends | id | friend_id | |----|-----------| | 1 | 4 | | 2 | 3 | | 3 | 2 | | 4 | 1 |

Packages | id | salary | |----|--------| | 1 | 10 | | 2 | 20 | | 3 | 30 | | 4 | 40 |

Output

Result | name | |------| | A | | B | | C |

💡 Solution (MySQL)

```
WITH cte (id, frnd_id, sal, frnd_sal) AS (
    SELECT
        Friends.id AS id,
        Friends.friend_id AS frnd_id,
        Packages_ID.salary AS sal,
        Packages_Friend_ID.salary AS frnd_sal
    FROM Friends
    INNER JOIN Packages AS Packages_ID
        ON Packages_ID.id = Friends.id
    INNER JOIN Packages AS Packages_Friend_ID
        ON Packages_Friend_ID.id = Friends.friend_id
)
SELECT Students.name
FROM Students
INNER JOIN cte
    ON cte.id = Students.id
    AND cte.frnd_sal > cte.sal
ORDER BY cte.frnd_sal ASC;
```

Understanding the Problem

We need to:

1. Find students whose best friends have higher salaries than them
2. Return these students' names in ascending order of their friends' salary

3. Work with three tables: Students (names), Friends (best friend relationships), and Packages (salary data)

Breaking Down the SQL Query

The solution uses a Common Table Expression (CTE) and multiple joins. Let's break it down:

```
WITH cte (id, frnd_id, sal, frnd_sal) AS (
    SELECT
        Friends.id AS id,
        Friends.friend_id AS frnd_id,
        Packages_ID.salary AS sal,
        Packages_Friend_ID.salary AS frnd_sal
    FROM Friends
    INNER JOIN Packages AS Packages_ID
        ON Packages_ID.id = Friends.id
    INNER JOIN Packages AS Packages_Friend_ID
        ON Packages_Friend_ID.id = Friends.friend_id
)
SELECT Students.name
FROM Students
INNER JOIN cte
    ON cte.id = Students.id
    AND cte.frnd_sal > cte.sal
ORDER BY cte.frnd_sal ASC;
```

1. The Common Table Expression (CTE)

```
WITH cte (id, frnd_id, sal, frnd_sal) AS (
    -- Query here
)
```

A CTE is like a temporary result set that exists only for this query. Think of it as creating a temporary table that we can reference later. We've named our CTE "cte" and defined the column names.

2. Inside the CTE: Gathering Salary Information

```
SELECT
    Friends.id AS id,
    Friends.friend_id AS frnd_id,
    Packages_ID.salary AS sal,
    Packages_Friend_ID.salary AS frnd_sal
FROM Friends
INNER JOIN Packages AS Packages_ID
    ON Packages_ID.id = Friends.id
INNER JOIN Packages AS Packages_Friend_ID
    ON Packages_Friend_ID.id = Friends.friend_id
```

This part is doing the core work:

- Starting with the Friends table to get friend relationships

- Joining with Packages table TWICE (with two different aliases):
  - First join (Packages_ID): Gets the student's OWN salary
  - Second join (Packages_Friend_ID): Gets the student's FRIEND'S salary
- The result is a dataset containing each student, their friend, and both salaries

3. The Main Query: Finding Students with Lower Salaries

```
SELECT Students.name
FROM Students
INNER JOIN cte
    ON cte.id = Students.id
    AND cte.frnd_sal > cte.sal
ORDER BY cte.frnd_sal ASC;
```

Now we join our CTE with the Students table to:

- Get the student names (since our CTE only has IDs)
- Filter for students where friend's salary > student's salary
- Sort the results by the friend's salary in ascending order

Walking Through an Example

Let's work through the example data step by step:

Students | id | name | |----|------| | 1 | A | | 2 | B | | 3 | C | | 4 | D |

Friends | id | friend_id | |----|-----------| | 1 | 4 | | 2 | 3 | | 3 | 2 | | 4 | 1 |

Packages | id | salary | |----|--------| | 1 | 10 | | 2 | 20 | | 3 | 30 | | 4 | 40 |

Step 1: Compute the CTE

After executing the CTE part, we'd get:

id	frnd_id	sal	frnd_sal
1	4	10	40
2	3	20	30
3	2	30	20
4	1	40	10

This gives us each student, their friend, the student's salary, and their friend's salary.

Step 2: Filter and Join with Students

Now we join this with Students and filter where frnd_sal > sal:

id	name	frnd_id	sal	frnd_sal
1	A	4	10	40
2	B	3	20	30

Student 3 (C) and Student 4 (D) are filtered out because their friends have lower salaries.

Step 3: Final Output

After selecting just the name and ordering by frnd_sal:

name
B
A

Wait, this doesn't match the expected output! The expected output shows:

name
A
B
C

Let's double-check our analysis:

- Student A (id=1): Salary=10, Friend=D (id=4), Friend's Salary=40 □ (Friend earns more)
- Student B (id=2): Salary=20, Friend=C (id=3), Friend's Salary=30 □ (Friend earns more)
- Student C (id=3): Salary=30, Friend=B (id=2), Friend's Salary=20 □ (Friend earns less)
- Student D (id=4): Salary=40, Friend=A (id=1), Friend's Salary=10 □ (Friend earns less)

So A and B should be in the output but not C. There might be an issue with the expected output provided in the problem statement.

Key Concepts for Beginners

1. **Common Table Expressions (CTEs)**: Temporary named result sets that exist for a single query
2. **Multiple Table Aliases**: We joined the Packages table twice with different aliases to get both salaries
3. **JOIN with Conditions**: The join to the CTE includes both the key match AND the salary comparison
4. **ORDER BY**: Sorting the results based on a specific column

Why This Approach Works Well

This solution is elegant because:

1. The CTE makes the complex logic more readable by breaking it into steps
2. Joining the Packages table twice avoids complex subqueries
3. The filtering happens in the JOIN condition, making the query more efficient

For beginners, remember that CTEs are powerful tools for breaking down complex queries into more manageable pieces. Think of them as creating temporary tables that help

simplify your logic.