

```

WITH
-- Step 1: Precompute the creator for each message
message_creators AS (
    SELECT
        m.m_messageid,
        m.m_creatorid
    FROM
        message m
),

-- Step 2: Precompute foreign likes for each message
foreign_likes AS (
    SELECT
        l.l_messageid AS message_id,
        COUNT(l.l_personid) AS foreign_like_count
    FROM
        likes l
    LEFT JOIN knows k ON l.l_personid = k.k_person2id -- Check if the
liker is in the "knows" table
    LEFT JOIN message_creators mc ON k.k_person1id = mc.m_creatorid
    WHERE
        mc.m_creatorid IS NULL -- Exclude likes from friends of the
creator
    GROUP BY
        l.l_messageid
),

-- Step 3: Precompute total likes for each message
total_likes AS (
    SELECT
        l_messageid AS message_id,
        COUNT(l_personid) AS total_like_count
    FROM
        likes
    GROUP BY
        l_messageid
)
SELECT
    m.m_messageid AS message_id,           -- The ID of the message
    tl.total_like_count,                   -- Total number of likes for the
message
    fl.foreign_like_count,                 -- Number of foreign likes for
the message
    COALESCE(fl.foreign_like_count * 1.0 / NULLIF(tl.total_like_count, 0),
0) AS foreign_like_ratio                 -- Ratio of foreign likes to
total likes

```

```

FROM
    message m
JOIN
    total_likes tl ON m.m_messageid = tl.message_id
LEFT JOIN
    foreign_likes fl ON m.m_messageid = fl.message_id
WHERE
    m.m_length > 100 -- Filter messages longer than
100 characters
    AND tl.total_like_count >= 20 -- Filter messages with at least
20 total likes
    AND COALESCE(fl.foreign_like_count, 0) >= tl.total_like_count / 2
-- Ensure at least half of the
likes are foreign
ORDER BY
    foreign_like_ratio DESC; -- Sort by the foreign-like
ratio in descending order

```

I want you to extend this query to include the full name of the first foreign person who liked the message and the time when they created the like. The attached schema file might be helpful to you.

Chat GPT Response

ChatGPT said:

ChatGPT

The new schema file appears to have the same structure as the previous one. Since the relevant details were identified earlier (e.g., `likes`, `person`, and `knows` tables), I will directly extend the query to include the full name of the first foreign liker and the time when the like was created. Here's the extended query:

sql

Copy code

```

WITH
    -- Step 1: Precompute the creator for each message
    message_creators AS (
        SELECT
            m.m_messageid,
            m.m_creatorid
        FROM
            message m
    ),

```

```

-- Step 2: Precompute foreign likes for each message
foreign_likes AS (
    SELECT
        l.l_messageid AS message_id,
        COUNT(l.l_personid) AS foreign_like_count
    FROM
        likes l
        LEFT JOIN knows k ON l.l_personid = k.k_person2id -- Check if
the liker is in the "knows" table
        LEFT JOIN message_creators mc ON k.k_person1id =
mc.m_creatorid
    WHERE
        mc.m_creatorid IS NULL -- Exclude likes from friends of
the creator
    GROUP BY
        l.l_messageid
),

```

```

-- Step 3: Precompute total likes for each message
total_likes AS (
    SELECT
        l_messageid AS message_id,
        COUNT(l_personid) AS total_like_count
    FROM
        likes
    GROUP BY
        l_messageid
),

```

```

-- Step 4: Get the first foreign liker for each message
first_foreign_likers AS (
    SELECT DISTINCT ON (l.l_messageid)
        l.l_messageid AS message_id,
        l.l_personid AS person_id,
        l.l_creationdate AS like_creation_date
    FROM
        likes l

```

```

        LEFT JOIN knows k ON l.l_personid = k.k_person2id
        LEFT JOIN message_creators mc ON k.k_person1id =
mc.m_creatorid
        WHERE
            mc.m_creatorid IS NULL -- Exclude likes from friends of
the creator
        ORDER BY
            l.l_messageid, l.l_creationdate ASC -- Get the earliest
like for each message
    )
SELECT
    m.m_messageid AS message_id,          -- The ID of the message
    tl.total_like_count,                  -- Total number of likes for
the message
    fl.foreign_like_count,                -- Number of foreign likes
for the message
    COALESCE(fl.foreign_like_count * 1.0 / NULLIF(tl.total_like_count,
0), 0) AS foreign_like_ratio,
                                           -- Ratio of foreign likes to
total likes
    CONCAT(p.p_firstname, ' ', p.p_lastname) AS
first_foreign_liker_name, -- Full name of the first foreign liker
    ffl.like_creation_date                -- Time when the first
foreign like was created
FROM
    message m
JOIN
    total_likes tl ON m.m_messageid = tl.message_id
LEFT JOIN
    foreign_likes fl ON m.m_messageid = fl.message_id
LEFT JOIN
    first_foreign_likers ffl ON m.m_messageid = ffl.message_id
LEFT JOIN
    person p ON ffl.person_id = p.p_personid
WHERE
    m.m_length > 100                    -- Filter messages longer
than 100 characters

```

```
        AND t1.total_like_count >= 20          -- Filter messages with at
least 20 total likes
        AND COALESCE(f1.foreign_like_count, 0) >= t1.total_like_count / 2
                                                -- Ensure at least half of
the likes are foreign
ORDER BY
    foreign_like_ratio DESC;                  -- Sort by the foreign-like
ratio in descending order
```

Explanation of Changes:

1. **Added `first_foreign_likers` CTE:** This identifies the first foreign liker for each message by sorting likes based on their creation date.
2. **Joined `person` table:** Fetches the full name of the first foreign liker.
3. **Included `like_creation_date`:** Provides the timestamp of when the first foreign like occurred.

This query fulfills the new requirement and builds on the existing logic seamlessly.