

**Du:**

I want to create an SQL Query. Could you help me? This is the query in natural language: "Find all messages longer than 100 characters that have been liked by at least 20 people. Only consider messages where at least half of the likes are from people that are foreign to the message's creator, with a foreigner being a person that neither the creator nor any of their friends know. For each message, return the message's id, the total number of likes and the number of foreign likes. Sort the returned messages by the ratio of foreign likes from highest to lowest." This is my database schema:

//Inserted schema here

**ChatGPT:**

ChatGPT

To create the SQL query you described, we need to address multiple parts step by step. Let's break down the requirements and map them onto the schema you've provided:

### Key Parts of the Query

1. **Messages longer than 100 characters** — We can use the message table and check the m\_length field.
2. **Liked by at least 20 people** — We'll use the likes table, counting the number of likes per message.
3. **Foreign likes definition** — A like is foreign if the person who liked the message is not known by the creator of the message nor by any of their friends.
4. **Return required fields** — Message ID, total number of likes, number of foreign likes.
5. **Sort by ratio of foreign likes** — This involves calculating the ratio and sorting by it.

### Table References

- message: Contains the message details (m\_messageid, m\_length, m\_creatorid).
- likes: Tracks who liked which message (l\_personid, l\_messageid).
- knows: Represents friendships (k\_person1id, k\_person2id).
- person: For person information (p\_personid).

### Solution Steps

1. **Filter messages by length:** Only consider messages with m\_length > 100.
2. **Count likes:** Group by m\_messageid and filter for messages with at least 20 likes.
3. **Identify foreign likes:**
  - Use the knows table to find all people known by the message creator.
  - Consider a like as foreign if the liker is not in the creator's social circle.
4. **Calculate required metrics and sort.**

### SQL Query

Here's the SQL query that implements the described logic:

sql

Code kopieren

WITH KnownPeople AS (

-- Get all known people for each message creator (direct friends and friends of friends)

SELECT

k1.k\_person1id AS creator,

```

    k2.k_person2id AS known_person
FROM
    knows k1
JOIN
    knows k2 ON k1.k_person2id = k2.k_person1id

UNION

SELECT
    k1.k_person2id AS creator,
    k1.k_person1id AS known_person
FROM
    knows k1
),
MessageLikes AS (
    -- Count total likes per message and filter messages with more than 20 likes
    SELECT
        l.l_messageid,
        COUNT(l.l_personid) AS total_likes
    FROM
        likes l
    GROUP BY
        l.l_messageid
    HAVING
        COUNT(l.l_personid) >= 20
),
ForeignLikes AS (
    -- Count foreign likes per message
    SELECT
        l.l_messageid,
        COUNT(l.l_personid) AS foreign_likes
    FROM
        likes l
    JOIN
        message m ON l.l_messageid = m.m_messageid
    LEFT JOIN
        KnownPeople kp ON kp.creator = m.m_creatorid AND kp.known_person =
l.l_personid
    WHERE
        kp.known_person IS NULL
    GROUP BY
        l.l_messageid
),
FilteredMessages AS (
    -- Filter messages longer than 100 characters and join with likes data
    SELECT
        m.m_messageid,
        ml.total_likes,

```

```

        fl.foreign_likes
FROM
    message m
JOIN
    MessageLikes ml ON m.m_messageid = ml.l_messageid
JOIN
    ForeignLikes fl ON m.m_messageid = fl.l_messageid
WHERE
    m.m_length > 100
)
-- Final result: Calculate the ratio of foreign likes and sort
SELECT
    m_messageid,
    total_likes,
    foreign_likes,
    CAST(foreign_likes AS DECIMAL) / total_likes AS foreign_like_ratio
FROM
    FilteredMessages
ORDER BY
    foreign_like_ratio DESC;

```

#### Explanation

1. **KnownPeople CTE:** Identifies all people known by the creator of the message, either directly or through friends.
2. **MessageLikes CTE:** Aggregates the total likes per message and filters out messages with fewer than 20 likes.
3. **ForeignLikes CTE:** Counts likes from users who are not in the creator's social circle.
4. **FilteredMessages CTE:** Applies the message length filter and combines all necessary metrics.
5. **Final SELECT:** Computes the foreign like ratio and sorts the results in descending order.

#### Notes

- The CAST in the final query ensures decimal division, avoiding integer truncation.
- Using LEFT JOIN with KnownPeople and checking for IS NULL effectively identifies foreign likes.

#### Du:

The Query is almost correct. However, the number of foreign likes needs to be at least half the amount of the number of likes. Could you generate the query again and fix this, please?

#### ChatGPT:

ChatGPT

Good catch! We need to ensure that the number of foreign likes is at least half of the total likes for each message. Let's adjust the query to incorporate this condition.

#### Updated SQL Query

sql

Code kopieren

```

WITH KnownPeople AS (
  -- Get all known people for each message creator (direct friends and friends of friends)
  SELECT
    k1.k_person1id AS creator,
    k2.k_person2id AS known_person
  FROM
    knows k1
  JOIN
    knows k2 ON k1.k_person2id = k2.k_person1id

  UNION

  SELECT
    k1.k_person2id AS creator,
    k1.k_person1id AS known_person
  FROM
    knows k1
),
MessageLikes AS (
  -- Count total likes per message and filter messages with more than 20 likes
  SELECT
    l.l_messageid,
    COUNT(l.l_personid) AS total_likes
  FROM
    likes l
  GROUP BY
    l.l_messageid
  HAVING
    COUNT(l.l_personid) >= 20
),
ForeignLikes AS (
  -- Count foreign likes per message
  SELECT
    l.l_messageid,
    COUNT(l.l_personid) AS foreign_likes
  FROM
    likes l
  JOIN
    message m ON l.l_messageid = m.m_messageid
  LEFT JOIN
    KnownPeople kp ON kp.creator = m.m_creatorid AND kp.known_person =
l.l_personid
  WHERE
    kp.known_person IS NULL
  GROUP BY
    l.l_messageid
),
FilteredMessages AS (

```

-- Filter messages longer than 100 characters, join with likes data, and ensure at least half of the likes are foreign

```
SELECT
    m.m_messageid,
    ml.total_likes,
    fl.foreign_likes
FROM
    message m
JOIN
    MessageLikes ml ON m.m_messageid = ml.l_messageid
JOIN
    ForeignLikes fl ON m.m_messageid = fl.l_messageid
WHERE
    m.m_length > 100
    AND fl.foreign_likes >= (ml.total_likes / 2)
)
-- Final result: Calculate the ratio of foreign likes and sort
SELECT
    m_messageid,
    total_likes,
    foreign_likes,
    CAST(foreign_likes AS DECIMAL) / total_likes AS foreign_like_ratio
FROM
    FilteredMessages
ORDER BY
    foreign_like_ratio DESC;
```

### Changes Made

1. **Foreign Likes Condition:** Added `fl.foreign_likes >= (ml.total_likes / 2)` in the `FilteredMessages` CTE to ensure that the number of foreign likes is at least half of the total likes.
2. **Other Parts Remain Unchanged:** The query still filters by message length, counts total likes and foreign likes, and sorts by the ratio of foreign likes.

### Explanation

- **ForeignLikes CTE:** Counts the number of foreign likes per message.
- **FilteredMessages CTE:** Filters for messages that have at least half of their likes as foreign likes.
- **Final SELECT:** Returns the message ID, total likes, foreign likes, and the ratio of foreign likes, sorted by this ratio in descending order.

This query should now fully meet your requirements. Let me know if you need any further adjustments!

**Du:**

Thank you! Now the query shall be extended as follows: “For each message, also return the full name of the first foreign person who liked the message and the time when they created the like.” Please generate the new query.

**ChatGPT:**

ChatGPT

To extend the query based on your new requirements, we need to include the following additional information for each message:

1. **The full name of the first foreign person who liked the message.**
2. **The timestamp of when this like was created.**

#### Changes to Implement

- We will use the person table to fetch the full name of the liker.
- We will join the likes table to retrieve the timestamp of the like.
- We need to identify the *first* foreign like for each message. This can be done using a ROW\_NUMBER() window function to rank likes by timestamp.

#### Updated SQL Query

sql

Code kopieren

```
WITH KnownPeople AS (
  -- Get all known people for each message creator (direct friends and friends of friends)
  SELECT
    k1.k_person1id AS creator,
    k2.k_person2id AS known_person
  FROM
    knows k1
  JOIN
    knows k2 ON k1.k_person2id = k2.k_person1id

  UNION

  SELECT
    k1.k_person2id AS creator,
    k1.k_person1id AS known_person
  FROM
    knows k1
),
MessageLikes AS (
  -- Count total likes per message and filter messages with at least 20 likes
  SELECT
    l.l_messageid,
    COUNT(l.l_personid) AS total_likes
  FROM
    likes l
  GROUP BY
    l.l_messageid
  HAVING
    COUNT(l.l_personid) >= 20
),
ForeignLikes AS (
  -- Identify foreign likes and count them per message
  SELECT
    l.l_messageid,
    l.l_personid,
    l.l_creationdate,
```

```

        COUNT(l.l_personid) OVER (PARTITION BY l.l_messageid) AS foreign_likes
FROM
    likes l
JOIN
    message m ON l.l_messageid = m.m_messageid
LEFT JOIN
    KnownPeople kp ON kp.creator = m.m_creatorid AND kp.known_person =
l.l_personid
WHERE
    kp.known_person IS NULL
),
FilteredMessages AS (
    -- Filter messages longer than 100 characters and ensure at least half of the likes are
foreign
    SELECT
        m.m_messageid,
        ml.total_likes,
        COUNT(fl.l_personid) AS foreign_likes
FROM
    message m
JOIN
    MessageLikes ml ON m.m_messageid = ml.l_messageid
JOIN
    ForeignLikes fl ON m.m_messageid = fl.l_messageid
WHERE
    m.m_length > 100
GROUP BY
    m.m_messageid, ml.total_likes
HAVING
    COUNT(fl.l_personid) >= (ml.total_likes / 2)
),
FirstForeignLike AS (
    -- Get the first foreign like for each message using ROW_NUMBER
    SELECT
        fl.l_messageid,
        fl.l_personid,
        fl.l_creationdate,
        ROW_NUMBER() OVER (PARTITION BY fl.l_messageid ORDER BY fl.l_creationdate)
AS rn
FROM
    ForeignLikes fl
),
FirstForeignLikerDetails AS (
    -- Retrieve the details of the first foreign liker
    SELECT
        ffl.l_messageid,
        p.p_firstname || ' ' || p.p_lastname AS full_name,
        ffl.l_creationdate AS like_creation_time

```

```

FROM
    FirstForeignLike ffl
JOIN
    person p ON ffl.l_personid = p.p_personid
WHERE
    ffl.rn = 1
)
-- Final result: Return message details along with the first foreign liker information
SELECT
    fm.m_messageid,
    fm.total_likes,
    fm.foreign_likes,
    CAST(fm.foreign_likes AS DECIMAL) / fm.total_likes AS foreign_like_ratio,
    ffl.full_name AS first_foreign_liker,
    ffl.like_creation_time
FROM
    FilteredMessages fm
JOIN
    FirstForeignLikerDetails ffl ON fm.m_messageid = ffl.l_messageid
ORDER BY
    foreign_like_ratio DESC;

```

### Explanation

1. **KnownPeople CTE:** Identifies all known people for each message creator (direct friends and friends of friends).
2. **MessageLikes CTE:** Aggregates total likes per message and filters messages with at least 20 likes.
3. **ForeignLikes CTE:** Identifies likes from people who are not in the creator's social circle.
4. **FilteredMessages CTE:** Filters messages based on length and ensures that at least half of the likes are foreign.
5. **FirstForeignLike CTE:** Uses ROW\_NUMBER() to find the first foreign like per message, based on the creation date.
6. **FirstForeignLikerDetails CTE:** Joins the first foreign like with the person table to get the full name and like creation time.
7. **Final SELECT:** Returns the message ID, total likes, foreign likes, foreign like ratio, full name of the first foreign liker, and the like creation time, sorted by the foreign like ratio.

This query should now satisfy all the requirements. Let me know if you need any more adjustments!