

# SQL Query for Foreign Likes



I have a database with the following schema:

```
"create table post (
  m_messageid bigint not null,
  m_ps_imagefile varchar,
  m_creationdate timestamp with time zone not null,
  m_locationip varchar not null,
  m_browserused varchar not null,
  m_ps_language varchar,
  m_content text,
  m_length int not null,
  m_creatorid bigint not null,
  m_ps_forumid bigint,
  m_locationid bigint not null
);
```

```
create table comment (
  m_messageid bigint not null,
  m_creationdate timestamp with time zone not null,
  m_locationip varchar not null,
  m_browserused varchar not null,
  m_content text,
  m_length int not null,
  m_creatorid bigint not null,
  m_locationid bigint not null,
  m_replyof_post bigint,
  m_replyof_comment bigint
);
```

```
create table forum (
  f_forumid bigint not null,
  f_title varchar not null,
  f_creationdate timestamp with time zone not null,
  f_moderatorid bigint not null
);
```

```
create table forum_person (
  fp_forumid bigint not null,
  fp_personid bigint not null,
  fp_joindate timestamp with time zone not null
);
```

```
create table forum_tag (
  ft_forumid bigint not null,
  ft_tagid bigint not null
```

```
);

create table organisation (
  o_organisationid bigint not null,
  o_type varchar not null,
  o_name varchar not null,
  o_url varchar not null,
  o_placeid bigint not null
);

create table person (
  p_personid bigint not null,
  p_firstname varchar not null,
  p_lastname varchar not null,
  p_gender varchar not null,
  p_birthday date not null,
  p_creationdate timestamp with time zone not null,
  p_locationip varchar not null,
  p_browserused varchar not null,
  p_placeid bigint not null
);

create table person_email (
  pe_personid bigint not null,
  pe_email varchar not null
);

create table person_tag (
  pt_personid bigint not null,
  pt_tagid bigint not null
);

create table knows (
  k_person1id bigint not null,
  k_person2id bigint not null,
  k_creationdate timestamp with time zone not null
);

create table likes (
  l_personid bigint not null,
  l_messageid bigint not null,
  l_creationdate timestamp with time zone not null
);

create table person_language (
  plang_personid bigint not null,
  plang_language varchar not null
);

create table person_university (
```

```

    pu_personid bigint not null,
    pu_organisationid bigint not null,
    pu_classyear int not null
);

create table person_company (
    pc_personid bigint not null,
    pc_organisationid bigint not null,
    pc_workfrom int not null
);

create table place (
    pl_placeid bigint not null,
    pl_name varchar not null,
    pl_url varchar not null,
    pl_type varchar not null,
    pl_containerplaceid bigint -- null for continents
);

create table country (
    ctry_city bigint not null,
    ctry_name varchar not null
);

create table message (
    m_messageid bigint not null,
    m_ps_imagefile varchar,
    m_creationdate timestamp with time zone not null,
    m_locationip varchar not null,
    m_browserused varchar not null,
    m_ps_language varchar,
    m_content text,
    m_length int not null,
    m_creatorid bigint not null,
    m_locationid bigint not null,
    m_ps_forumid bigint,
    m_c_replyof bigint
);

create table message_tag (
    mt_messageid bigint not null,
    mt_tagid bigint not null
);

create table tagclass (
    tc_tagclassid bigint not null,
    tc_name varchar not null,
    tc_url varchar not null,
    tc_subclassoftagclassid bigint -- null for the root tagclass (Thing)
);

```

```
create table tag (
  t_tagid bigint not null,
  t_name varchar not null,
  t_url varchar not null,
  t_tagclassid bigint not null
);"
```

I needed to implement this query task:

"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."

For this I generated this query:

```
"WITH
  message_likes AS (
    SELECT
      l_messageid AS message_id,
      COUNT(*) AS total_likes
    FROM
      likes
    GROUP BY
      l_messageid
    HAVING
      COUNT(*) >= 20
  ),
  creator_friends AS (
    -- Direct friends of the message creator
    SELECT
      k_person2id AS friend_id,
      k_person1id AS creator_id
    FROM
      knows
  ),
  friends_of_friends AS (
    -- Friends of the creator's friends
    SELECT
      k.k_person2id AS person_id,
      cf.creator_id
    FROM
      knows k
    JOIN creator_friends cf ON k.k_person1id = cf.friend_id
  ),
  extended_network AS (
    -- Union of direct friends and friends of friends
    SELECT
      friend_id AS person_id,
      creator_id
    FROM
      creator_friends
    UNION
```

```

SELECT
    person_id,
    creator_id
FROM
    friends_of_friends
),
foreign_likes AS (
    SELECT
        l.l_messageid AS message_id,
        COUNT(*) AS foreign_likes
    FROM
        likes l
    JOIN message m ON l.l_messageid = m.m_messageid
    LEFT JOIN extended_network en ON en.person_id = l.l_personid
        AND en.creator_id = m.m_creatorid
    WHERE
        en.person_id IS NULL -- The liker is not in the extended network of the creator
    GROUP BY
        l.l_messageid
),
filtered_messages AS (
    SELECT
        ml.message_id,
        ml.total_likes,
        fl.foreign_likes
    FROM
        message_likes ml
    JOIN foreign_likes fl ON ml.message_id = fl.message_id
    WHERE
        fl.foreign_likes >= ml.total_likes / 2
)
SELECT
    fm.message_id,
    fm.total_likes,
    fm.foreign_likes
FROM
    filtered_messages fm
JOIN message m ON fm.message_id = m.m_messageid
WHERE
    m.m_length > 100
ORDER BY
    (fm.foreign_likes::decimal / fm.total_likes) DESC;"

```

now I need to expand it to also cover this following task;  
 "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"  
 Expand the given query



Here is the expanded query to include the additional requirement of returning the full name of the first foreign person who liked the message and the time when they created the like:

```
sql
```

```
WITH
  message_likes AS (
    SELECT
      l_messageid AS message_id,
      COUNT(*) AS total_likes
    FROM
      likes
    GROUP BY
      l_messageid
    HAVING
      COUNT(*) >= 20
  ),
  creator_friends AS (
    -- Direct friends of the message creator
    SELECT
      k_person2id AS friend_id,
      k_person1id AS creator_id
    FROM
      knows
  ),
  friends_of_friends AS (
    -- Friends of the creator's friends
    SELECT
      k.k_person2id AS person_id,
      cf.creator_id
    FROM
      knows k
    JOIN creator_friends cf ON k.k_person1id = cf.friend_id
  ),
  extended_network AS (
    -- Union of direct friends and friends of friends
    SELECT
      friend_id AS person_id,
      creator_id
    FROM
      creator_friends
    UNION
    SELECT
      person_id,
      creator_id
    FROM
      friends_of_friends
  ),
  foreign_likes AS (
    SELECT
      l.l_messageid AS message_id,
      COUNT(*) AS foreign_likes,
      MIN(l.l_creationdate) AS first_foreign_like_time,
      MIN(p.p_firstname || ' ' || p.p_lastname) AS first_foreign_liker_name
    FROM
      likes l
    JOIN message m ON l.l_messageid = m.m_messageid
    LEFT JOIN extended_network en ON en.person_id = l.l_personid
      AND en.creator_id = m.m_creatorid
    JOIN person p ON l.l_personid = p.p_personid
    WHERE
      en.person_id IS NULL -- The liker is not in the extended network of the creator
    GROUP BY
      l.l_messageid
  ),
  filtered_messages AS (
    SELECT
      ml.message_id,
      ml.total_likes,
      fl.foreign_likes,
      fl.first_foreign_like_time,
      fl.first_foreign_liker_name
```

```

FROM
    message_likes ml
JOIN foreign_likes fl ON ml.message_id = fl.message_id
WHERE
    fl.foreign_likes >= ml.total_likes / 2
)
SELECT
    fm.message_id,
    fm.total_likes,
    fm.foreign_likes,
    fm.first_foreign_liker_name,
    fm.first_foreign_like_time
FROM
    filtered_messages fm
JOIN message m ON fm.message_id = m.m_messageid
WHERE
    m.m_length > 100
ORDER BY
    (fm.foreign_likes::decimal / fm.total_likes) DESC;

```

## Explanation of Additions:

### 1. `foreign\_likes` CTE:

- Added two new fields:
  - `MIN(l.l\_creationdate)` to capture the timestamp of the first foreign like.
  - `MIN(p.p\_firstname || ' ' || p.p\_lastname)` to get the full name of the first foreign liker.

### 2. `filtered\_messages` CTE:

- Pass through `first\_foreign\_like\_time` and `first\_foreign\_liker\_name` for each message.

### 3. Final SELECT:

- Include `fm.first\_foreign\_liker\_name` and `fm.first\_foreign\_like\_time` in the final output.

This will fulfill the requirement of returning the requested information for each message.



I think you are giving us the first person in alphabetical order instead of in chronological order



You are correct! My earlier implementation mistakenly used `MIN` for the foreign liker's name, which would indeed return the name in alphabetical order instead of based on the creation time of the like. Here's the corrected query that ensures the first foreign person is identified chronologically:

```

sql

WITH
    message_likes AS (
        SELECT
            l_messageid AS message_id,
            COUNT(*) AS total_likes
        FROM
            likes
        GROUP BY
            l_messageid
        HAVING
            COUNT(*) >= 20
    )

```

```

),
creator_friends AS (
  -- Direct friends of the message creator
  SELECT
    k_person2id AS friend_id,
    k_person1id AS creator_id
  FROM
    knows
),
friends_of_friends AS (
  -- Friends of the creator's friends
  SELECT
    k.k_person2id AS person_id,
    cf.creator_id
  FROM
    knows k
  JOIN creator_friends cf ON k.k_person1id = cf.friend_id
),
extended_network AS (
  -- Union of direct friends and friends of friends
  SELECT
    friend_id AS person_id,
    creator_id
  FROM
    creator_friends
  UNION
  SELECT
    person_id,
    creator_id
  FROM
    friends_of_friends
),
foreign_likes AS (
  SELECT
    l.l_messageid AS message_id,
    COUNT(*) AS foreign_likes
  FROM
    likes l
  JOIN message m ON l.l_messageid = m.m_messageid
  LEFT JOIN extended_network en ON en.person_id = l.l_personid
  AND en.creator_id = m.m_creatorid
  WHERE
    en.person_id IS NULL -- The liker is not in the extended network of the creator
  GROUP BY
    l.l_messageid
),
first_foreign_like_details AS (
  SELECT
    l.l_messageid AS message_id,
    l.l_creationdate AS like_time,
    p.p_firstname || ' ' || p.p_lastname AS liker_name,
    ROW_NUMBER() OVER (PARTITION BY l.l_messageid ORDER BY l.l_creationdate ASC) AS
row_num
  FROM
    likes l
  JOIN message m ON l.l_messageid = m.m_messageid
  LEFT JOIN extended_network en ON en.person_id = l.l_personid
  AND en.creator_id = m.m_creatorid
  JOIN person p ON l.l_personid = p.p_personid
  WHERE
    en.person_id IS NULL -- The liker is not in the extended network of the creator
),
first_foreign_likes AS (
  SELECT
    message_id,
    liker_name,
    like_time
  FROM
    first_foreign_like_details
  WHERE
    row_num = 1
),
filtered_messages AS (
  SELECT
    ml.message_id,

```



```

        ml.total_likes,
        fl.foreign_likes,
        ffl.liker_name AS first_foreign_liker_name,
        ffl.like_time AS first_foreign_like_time
    FROM
        message_likes ml
    JOIN foreign_likes fl ON ml.message_id = fl.message_id
    JOIN first_foreign_likes ffl ON ml.message_id = ffl.message_id
    WHERE
        fl.foreign_likes >= ml.total_likes / 2
)
SELECT
    fm.message_id,
    fm.total_likes,
    fm.foreign_likes,
    fm.first_foreign_liker_name,
    fm.first_foreign_like_time
FROM
    filtered_messages fm
JOIN message m ON fm.message_id = m.m_messageid
WHERE
    m.m_length > 100
ORDER BY
    (fm.foreign_likes::decimal / fm.total_likes) DESC;

```

## Key Adjustments:

### 1. `first\_foreign\_like\_details` CTE:

- Use `ROW\_NUMBER()` with `PARTITION BY l.l\_messageid ORDER BY l.l\_creationdate ASC` to assign a unique rank to each foreign like per message based on creation time.
- This ensures the first liker chronologically is identified.

### 2. `first\_foreign\_likes` CTE:

- Filter the `ROW\_NUMBER()` result to select only the first row (`row\_num = 1`) for each message, which corresponds to the earliest foreign like.

### 3. Integration with `filtered\_messages`:

- Join `first\_foreign\_likes` with `filtered\_messages` to include the earliest foreign liker's name and the like's creation time.

This updated query ensures that the first foreign liker is identified chronologically, addressing the earlier issue.