

**Du:**

Now for this Query:

```
SELECT like_count.m_messageid, total_likes, foreign_likes,
first_foreign_person.p_firstname || ' ' || first_foreign_person.p_lastname AS
first_foreign_person_name, first_foreign_like_time AS first_foreign_like_time FROM
(SELECT COUNT(*) AS total_likes, m_messageid FROM (SELECT * FROM message
WHERE m_length > 100) messages INNER JOIN likes ON messages.m_messageid =
likes.l_messageid GROUP BY m_messageid) AS like_count INNER JOIN (SELECT
DISTINCT ON (m_messageid) m_messageid, COUNT(*) OVER (PARTITION BY
m_messageid) AS foreign_likes, l_personid AS first_foreign_person_id, l_creationdate AS
first_foreign_like_time FROM (SELECT * FROM message WHERE m_length > 100)
messages INNER JOIN likes ON messages.m_messageid = likes.l_messageid WHERE
l_personid NOT IN (SELECT k_person2id FROM knows WHERE k_person1id =
m_creatorid) AND l_personid NOT IN (SELECT k2.k_person2id FROM knows k1, knows
k2 WHERE k1.k_person1id = m_creatorid AND k1.k_person2id = k2.k_person1id) ORDER
BY m_messageid, l_creationdate) AS foreign_like_count ON like_count.m_messageid =
foreign_like_count.m_messageid INNER JOIN person AS first_foreign_person ON
foreign_like_count.first_foreign_person_id = first_foreign_person.p_personid WHERE
like_count.total_likes >= 20 AND foreign_likes >= total_likes / 2 ORDER BY ((100 *
foreign_likes) / (total_likes)) DESC;
```

Give me an execution plan exactly like the one I would receive when putting EXPLAIN in the beginning and executing the query with postgresql.

As a Reminder here is the database Schema again:

```
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 );
```

## ChatGPT:

ChatGPT

Here's an execution plan simulating how PostgreSQL would likely interpret and execute this query. This is an estimate and may vary depending on the database's statistics and optimization strategies.

plaintext

Code kopieren

```
Sort (cost=#####.##..#####.## rows=### width=##) Sort Key: (((100 * foreign_like_count.foreign_likes) / like_count.total_likes)) DESC -> Nested Loop (cost=#####.##..#####.## rows=### width=##) Join Filter: (like_count.m_messageid = foreign_like_count.m_messageid) -> Hash Join (cost=#####.##..#####.## rows=### width=##) Hash Cond: (foreign_like_count.first_foreign_person_id = first_foreign_person.p_personid) -> Subquery Scan on foreign_like_count (cost=#####.##..#####.## rows=### width=##) -> WindowAgg (cost=#####.##..#####.## rows=### width=##) -> Sort (cost=#####.##..#####.## rows=### width=##) Sort Key: messages.m_messageid, likes.l_creationdate -> Hash Join (cost=#####.##..#####.## rows=### width=##) Hash Cond: (likes.l_messageid = messages.m_messageid) -> Seq Scan on likes (cost=#####.##..#####.## rows=### width=##) -> Hash (cost=#####.##..#####.## rows=### width=##) -> Seq Scan on messages (cost=#####.##..#####.## rows=### width=##) Filter: (m_length > 100) -> Hash (cost=#####.##..#####.## rows=### width=##) -> Seq Scan on person_first_foreign_person (cost=#####.##..#####.## rows=### width=##) -> Materialize (cost=#####.##..#####.## rows=### width=##) -> Subquery Scan on like_count (cost=#####.##..#####.## rows=### width=##) -> HashAggregate (cost=#####.##..#####.## rows=### width=##) Group Key:
```

messages.m\_messageid -> Hash Join (cost=#####.##..#####.## rows=###  
 width=##) Hash Cond: (likes.l\_messageid = messages.m\_messageid) -> Seq Scan on  
 likes (cost=#####.##..#####.## rows=### width=##) -> Hash  
 (cost=#####.##..#####.## rows=### width=##) -> Seq Scan on messages  
 (cost=#####.##..#####.## rows=### width=##) Filter: (m\_length > 100) Filter:  
 ((like\_count.total\_likes >= 20) AND (foreign\_like\_count.foreign\_likes >=  
 (like\_count.total\_likes / 2)))

Explanation of execution steps:

1. **Sequential Scans on messages and likes:** The query starts with sequential scans on messages (filtered to only include messages with m\_length > 100) and likes. Hash joins are then used to link likes entries to messages.
2. **Aggregation for total\_likes:** After joining, a HashAggregate step computes the total\_likes per message.
3. **Foreign Like Filter:** A filtered join excludes "known" persons using the knows table relationships, with a WindowAgg step to find the first foreign\_like\_time.
4. **Hash Joins with person:** The query retrieves first\_foreign\_person information with a hash join, linking to person table data.
5. **Final Sorting:** The result is sorted by the percentage of foreign likes ((100 \* foreign\_likes) / total\_likes DESC)