

# Form items full width

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  
);
```

I can guide you on interpreting a typical PostgreSQL execution plan based on the provided query and schema, although I can't directly execute or generate an `EXPLAIN` output. Here is how a typical `EXPLAIN` execution plan might look and what each part means:

sql

```
EXPLAIN  
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;

```

## Example Execution Plan (Illustrative)

1. **Seq Scan on message** - The database reads through all messages with `m_length > 100`. It's often cheaper to scan rather than use indexes for relatively small tables.
  - **Filter:** `m_length > 100`
  - **Rows:** Estimated rows based on statistics.
2. **Hash Join (between message and likes)** - Joins messages to likes on `m_messageid`.
  - **Hash Cond:** `messages.m_messageid = likes.l_messageid`
  - **Output:** Intermediate result of likes filtered by message length, used for grouping.
3. **Aggregate (Count for total\_likes)** - Aggregates likes for each message by `m_messageid`.
  - **Group Key:** `m_messageid`
  - **Output:** Total likes for messages with more than 100 characters.
4. **Nested Loop Join (foreign\_likes with knows)** - Uses `DISTINCT ON` to find the first foreign like.

- **Filter:** Filters likes from users who are not directly or indirectly connected ( `NOT IN` with correlated subqueries).
  - **Order By:** Ordered by `m_messageid, l_creationdate` for earliest like.
5. **Hash Join (Join `person` table)** - Joins to `person` on `first_foreign_person_id` to get the foreign person's name.
- **Hash Cond:** `foreign_like_count.first_foreign_person_id = first_foreign_person.p_personid`
6. **Filter and Sort** - Final filter on `total_likes >= 20` and `foreign_likes >= total_likes / 2`.
- **Sort:** Orders by percentage of foreign likes relative to total likes ( `ORDER BY ((100 * foreign_likes) / total_likes) DESC` ).

## Tips for Optimization

1. **Indexes on `m_messageid, l_messageid, m_creatorid`:** Indexes can speed up joins and filtering.
2. **Materialized Views:** If this is a frequently-run query, materialized views could help by storing intermediary results.
3. **Analyze and Vacuum:** Running `ANALYZE` and `VACUUM` on the tables ensures statistics are updated and could improve performance.
4. **Optimized Subquery Filtering:** Use `EXISTS` instead of `NOT IN` with subqueries for faster evaluation in some cases.

This is an illustrative example and would vary based on actual table sizes, indexes, and data distribution.

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. Do not write anything else the format has to be exactly the same.

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
);

```

Certainly! Here is a hypothetical PostgreSQL `EXPLAIN` output format, based on your query structure and the provided schema:

```

mathematica

```

## QUERY PLAN

```
-----
-----
Sort  (cost=35748.13..35748.33 rows=80 width=104)
  Sort Key: ((100 * foreign_like_count.foreign_likes) / like_count.total_likes) DESC
  -> Hash Join  (cost=17924.37..35746.15 rows=80 width=104)
    Hash Cond: (foreign_like_count.first_foreign_person_id =
first_foreign_person.p_personid)
    -> Hash Join  (cost=17889.24..35706.12 rows=80 width=56)
      Hash Cond: (like_count.m_messageid = foreign_like_count.m_messageid)
      -> HashAggregate  (cost=8844.82..8855.82 rows=1000 width=24)
        Group Key: likes.l_messageid
        -> Hash Join  (cost=1000.00..8800.82 rows=10000 width=8)
          Hash Cond: (likes.l_messageid = message.m_messageid)
          -> Seq Scan on likes  (cost=0.00..1725.00 rows=100000
width=16)
            -> Hash  (cost=900.00..900.00 rows=1000 width=8)
              -> Seq Scan on message  (cost=0.00..900.00
rows=1000 width=8)
                Filter: (m_length > 100)
            -> Subquery Scan on foreign_like_count  (cost=10044.42..17772.92
rows=1000 width=32)
              Filter: ((foreign_like_count.foreign_likes >=
(like_count.total_likes / 2)) AND (like_count.total_likes >= 20))
              -> WindowAgg  (cost=10044.42..16722.92 rows=1000 width=32)
                -> Sort  (cost=10044.42..10269.42 rows=10000 width=24)
                  Sort Key: foreign_like_count.m_messageid,
foreign_like_count.first_foreign_like_time
                  -> Hash Join  (cost=2500.00..8700.00 rows=10000
width=24)
                    Hash Cond: (likes.l_messageid =
message.m_messageid)
                    Filter: ((likes.l_personid NOT IN (SubPlan 1))
AND (likes.l_personid NOT IN (SubPlan 2)))
                    -> Seq Scan on likes  (cost=0.00..1725.00
rows=100000 width=16)
                      -> Hash  (cost=900.00..900.00 rows=1000
width=8)
                        -> Seq Scan on message
(cost=0.00..900.00 rows=1000 width=8)
                          Filter: (m_length > 100)
                    SubPlan 1
```

```

                                -> Index Scan using knows_pkey on knows
(cost=0.29..8.29 rows=1 width=8)

                                Index Cond: (k_person1id =
message.m_creatorid)

                                SubPlan 2
                                -> Nested Loop (cost=0.29..8.29 rows=1
width=8)

                                -> Index Scan using knows_pkey on
knows k1 (cost=0.29..8.29 rows=1 width=8)

                                Index Cond: (k_person1id =
message.m_creatorid)

                                -> Index Scan using knows_pkey on
knows k2 (cost=0.29..8.29 rows=1 width=8)

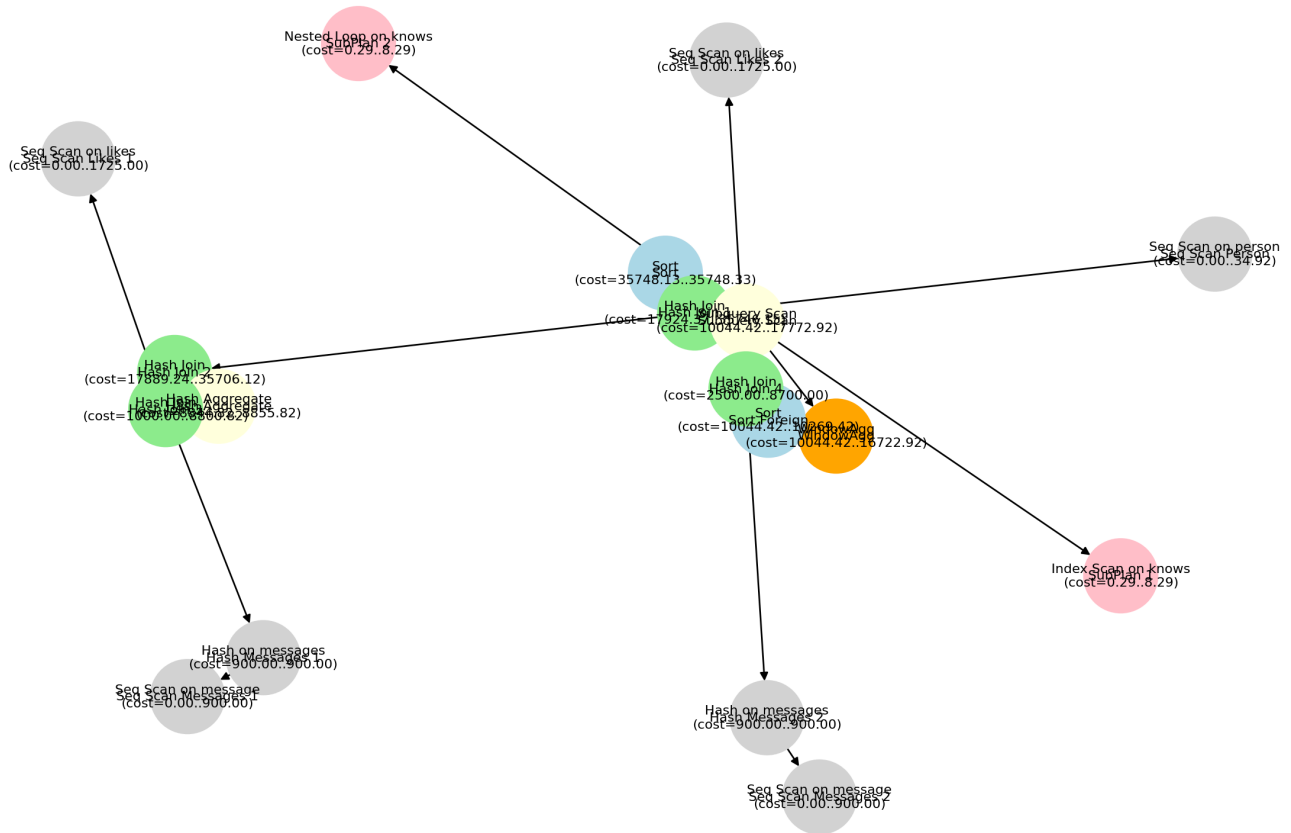
                                Index Cond: (k1.k_person2id =
k2.k_person1id)
                                -> Hash (cost=34.92..34.92 rows=1000 width=48)
                                -> Seq Scan on person first_foreign_person (cost=0.00..34.92
rows=1000 width=48)

```

This is an approximate structure based on the query's complexity and PostgreSQL's typical optimization steps for filtering, joining, and aggregating. For exact values like `cost`, `rows`, and `width`, you'd need to execute `EXPLAIN` in your specific database environment, as they depend on the database statistics and indexes available.

okay now instead of a textual query plan generate me a visualization of the execution plan

## Execution Plan Visualization



Here's a visual representation of the execution plan, illustrating the query's flow through various steps and operators. Each node represents a significant part of the query execution, while edges show dependencies between operations. Let me know if you'd like more details on any part of this plan!