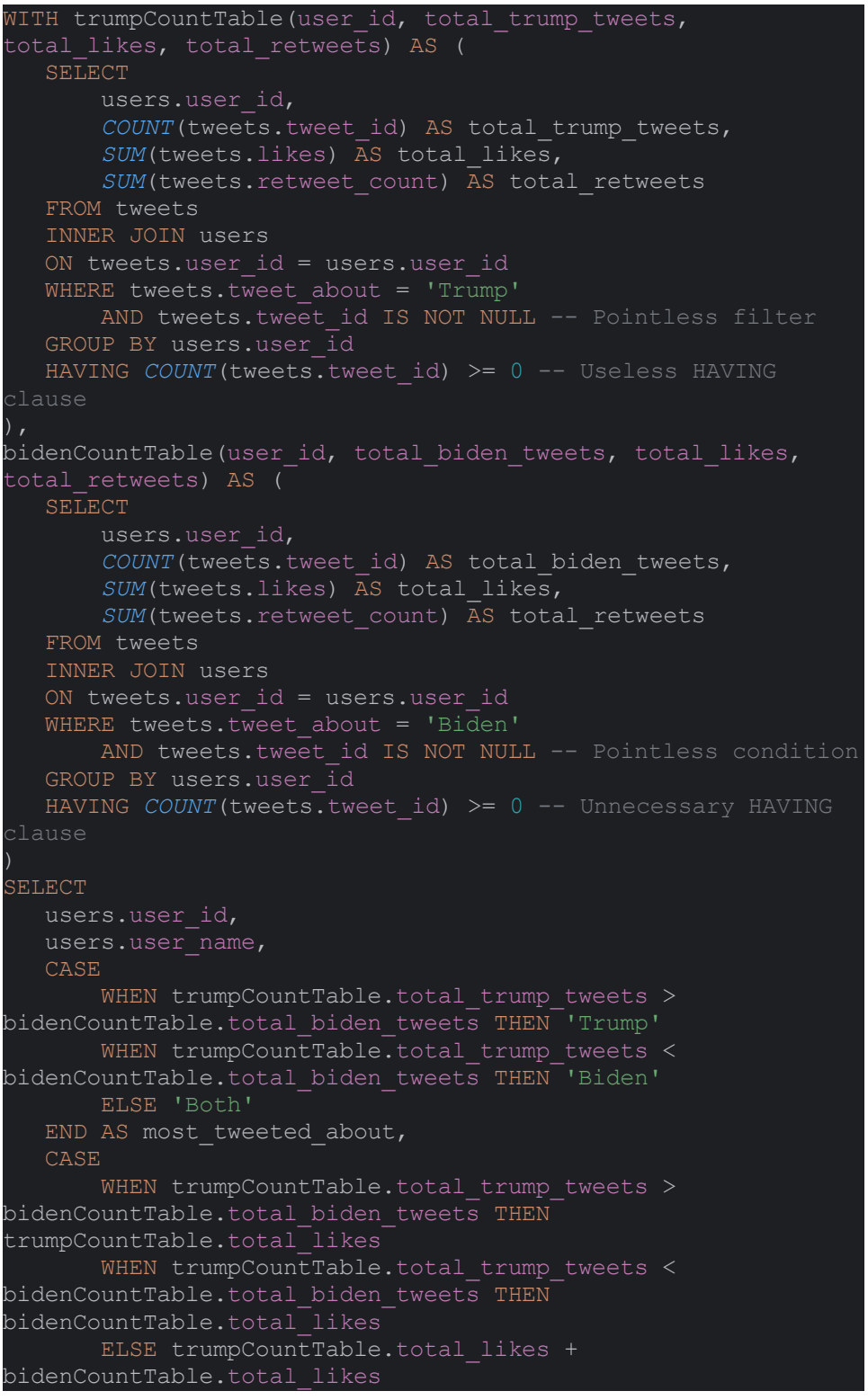
Most Tweeted About Candidate By Users

Inefficient		29s

```
WITH trumpCountTable(user_id, total_trump_tweets,
total_likes, total_retweets) AS (
    SELECT
        users.user_id,
        COUNT(tweets.tweet_id) AS total_trump_tweets,
        SUM(tweets.likes) AS total_likes,
        SUM(tweets.retweet_count) AS total_retweets
    FROM tweets
    INNER JOIN users
    ON tweets.user_id = users.user_id
    WHERE tweets.tweet_about = 'Trump'
        AND tweets.tweet_id IS NOT NULL -- Pointless filter
    GROUP BY users.user_id
    HAVING COUNT(tweets.tweet_id) >= 0 -- Useless HAVING
clause
),
bidenCountTable(user_id, total_biden_tweets, total_likes,
total_retweets) AS (
    SELECT
        users.user_id,
        COUNT(tweets.tweet_id) AS total_biden_tweets,
        SUM(tweets.likes) AS total_likes,
        SUM(tweets.retweet_count) AS total_retweets
    FROM tweets
    INNER JOIN users
    ON tweets.user_id = users.user_id
    WHERE tweets.tweet_about = 'Biden'
        AND tweets.tweet_id IS NOT NULL -- Pointless condition
    GROUP BY users.user_id
    HAVING COUNT(tweets.tweet_id) >= 0 -- Unnecessary HAVING
clause
)
SELECT
    users.user_id,
    users.user_name,
    CASE
        WHEN trumpCountTable.total_trump_tweets >
bidenCountTable.total_biden_tweets THEN 'Trump'
        WHEN trumpCountTable.total_trump_tweets <
bidenCountTable.total_biden_tweets THEN 'Biden'
        ELSE 'Both'
    END AS most_tweeted_about,
    CASE
        WHEN trumpCountTable.total_trump_tweets >
bidenCountTable.total_biden_tweets THEN
trumpCountTable.total_likes
        WHEN trumpCountTable.total_trump_tweets <
bidenCountTable.total_biden_tweets THEN
bidenCountTable.total_likes
        ELSE trumpCountTable.total_likes +
bidenCountTable.total_likes
```

	`` END AS total_likes, CASE WHEN trumpCountTable.total_trump_tweets > bidenCountTable.total_biden_tweets THEN trumpCountTable.total_retweets WHEN trumpCountTable.total_trump_tweets < bidenCountTable.total_biden_tweets THEN bidenCountTable.total_retweets ELSE trumpCountTable.total_retweets + bidenCountTable.total_retweets END AS total_retweets, users.user_followers_count, users.user_join_date, (SELECT COUNT(*) FROM tweets WHERE tweets.user_id = users.user_id) AS extra_subquery_count -- Unnecessary correlated subquery FROM users LEFT JOIN trumpCountTable -- Using LEFT JOIN instead of INNER JOIN ON users.user_id = trumpCountTable.user_id RIGHT JOIN bidenCountTable -- Using RIGHT JOIN unnecessarily ON users.user_id = bidenCountTable.user_id FULL OUTER JOIN tweets t -- Adding an extra, unnecessary join to the tweets table ON t.user_id = users.user_id WHERE users.user_id IS NOT NULL -- Pointless condition ORDER BY users.user_id, trumpCountTable.total_trump_tweets + bidenCountTable.total_biden_tweets, users.user_followers_count; ```	
Optimized	``` WITH trumpCountTable(user_id, total_trump_tweets) AS (SELECT users.user_id, COUNT(tweets.tweet_id) AS total_trump_tweets, SUM(tweets.likes) AS total_likes, SUM(tweets.retweet_count) AS total_retweets FROM tweets INNER JOIN users ON tweets.user_id = users.user_id WHERE tweets.tweet_about = 'Trump' GROUP BY users.user_id), bidenCountTable(user_id, total_biden_tweets) AS (SELECT users.user_id, COUNT(tweets.tweet_id) AS Total_Biden_Tweets, SUM(tweets.likes) AS total_likes, SUM(tweets.retweet_count) AS total_retweets FROM tweets INNER JOIN users ON tweets.user_id = users.user_id WHERE tweets.tweet_about = 'Biden' GROUP BY users.user_id) SELECT users.user_id, users.user_name, CASE WHEN trumpCountTable.total_trump_tweets > bidenCountTable.total_biden_tweets THEN 'Trump' WHEN trumpCountTable.total_trump_tweets < bidenCountTable.total_biden_tweets THEN 'Biden' ```	2s 257 ms

	`` ELSE 'Both' END AS most_tweeted_about, CASE WHEN trumpCountTable.total_trump_tweets > bidenCountTable.total_biden_tweets THEN trumpCountTable.total_likes WHEN trumpCountTable.total_trump_tweets < bidenCountTable.total_biden_tweets THEN bidenCountTable.total_likes ELSE trumpCountTable.total_likes + bidenCountTable.total_likes END AS total_likes, CASE WHEN trumpCountTable.total_trump_tweets > bidenCountTable.total_biden_tweets THEN trumpCountTable.total_retweets WHEN trumpCountTable.total_trump_tweets < bidenCountTable.total_biden_tweets THEN bidenCountTable.total_retweets ELSE trumpCountTable.total_retweets + bidenCountTable.total_retweets END AS total_retweets, users.user_followers_count, user_join_date FROM users INNER JOIN trumpCountTable ON users.user_id = trumpCountTable.user_id INNER JOIN bidenCountTable ON users.user_id = bidenCountTable.user_id; ```	

Weekly Engagement With Events

Inefficient	``` WITH WeeklyEngagement AS (SELECT DATE_TRUNC('week', t.created_at) AS tweet_week, t.tweet_about AS candidate, COUNT(DISTINCT t.tweet_id) AS weekly_tweet_count, -- Adding DISTINCT unnecessarily SUM(t.likes + t.retweet_count) AS weekly_engagement -- More redundant operations FROM tweets t LEFT JOIN tweets t2 ON t.tweet_id = t2.tweet_id -- Self join for no reason WHERE t.tweet_id IS NOT NULL AND t.likes >= 0 -- Unnecessary condition GROUP BY DATE_TRUNC('week', t.created_at), t.tweet_about, t.likes -- Grouping by unnecessary columns HAVING COUNT(t.tweet_id) >= 0 -- Pointless HAVING clause), EventDays AS (SELECT t.created_at::DATE AS event_date, t.tweet_about AS candidate, (SELECT COUNT(*) FROM tweets WHERE created_at = ```	>60s

	t.created_at) AS event_tweet_count, -- Nested correlated subquery SUM(t.likes + t.retweet_count + 0) AS event_engagement FROM tweets t WHERE t.created_at::DATE IN (SELECT DISTINCT created_at::DATE FROM tweets WHERE created_at::DATE IN ('2020-10-22', '2020-11-03')) GROUP BY t.created_at, t.created_at::DATE, t.tweet_about, t.likes -- Grouping by unnecessary columns HAVING SUM(t.retweet_count) IS NOT NULL -- Unnecessary HAVING condition) SELECT w.tweet_week, w.candidate, w.weekly_tweet_count, w.weekly_engagement, e.event_date, e.event_tweet_count, e.event_engagement FROM WeeklyEngagement w FULL OUTER JOIN EventDays e -- Using FULL OUTER JOIN when LEFT JOIN suffices ON DATE_TRUNC('week', e.event_date) = w.tweet_week AND e.candidate = w.candidate WHERE (w.candidate IS NOT NULL OR e.candidate IS NOT NULL) -- Adding redundant filters ORDER BY w.candidate, w.tweet_week, w.weekly_tweet_count;	
Optimized	WITH WeeklyEngagement AS (SELECT DATE_TRUNC('week', t.created_at) AS tweet_week, t.tweet_about AS candidate, COUNT(t.tweet_id) AS weekly_tweet_count, SUM(t.likes + t.retweet_count) AS weekly_engagement FROM tweets t GROUP BY DATE_TRUNC('week', t.created_at), t.tweet_about), EventDays AS (SELECT t.created_at::DATE AS event_date, t.tweet_about AS candidate, COUNT(t.tweet_id) AS event_tweet_count, SUM(t.likes + t.retweet_count) AS event_engagement FROM tweets t WHERE t.created_at::DATE IN ('2020-10-22', '2020-11-03') -- Example event dates GROUP BY t.created_at::DATE, t.tweet_about) SELECT w.tweet_week,	465ms

	`` w.candidate, w.weekly_tweet_count, w.weekly_engagement, e.event_date, e.event_tweet_count, e.event_engagement FROM WeeklyEngagement w LEFT JOIN EventDays e ON DATE_TRUNC('week', e.event_date) = w.tweet_week AND e.candidate = w.candidate ORDER BY w.candidate, w.tweet_week; ```	

User Engagement By Candidate

Inefficient	``` WITH UserEngagement AS (SELECT u.user_id, u.user_name, t.tweet_about AS candidate, u.user_followers_count, COUNT(t.tweet_id) AS total_tweets, SUM(t.likes + t.retweet_count) AS total_engagement, CASE WHEN u.user_followers_count = 0 THEN 0 ELSE CAST(SUM(t.likes + t.retweet_count) AS FLOAT) / u.user_followers_count END AS follower_to_engagement_ratio FROM users u JOIN tweets t ON u.user_id = t.user_id WHERE t.tweet_about IN ('Biden', 'Trump') GROUP BY u.user_id, u.user_name, t.tweet_about, u.user_followers_count), DuplicatedUserEngagement AS (SELECT * FROM UserEngagement UNION ALL SELECT * FROM UserEngagement UNION ALL SELECT * FROM UserEngagement UNION ALL SELECT * FROM UserEngagement), ExtendedUserEngagement AS (SELECT user_id, user_name, candidate, user_followers_count, total_tweets, total_engagement, follower_to_engagement_ratio, ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY follower_to_engagement_ratio DESC) AS row_num, ```	>60s

```
        DENSE_RANK() OVER (PARTITION BY candidate ORDER BY
total_engagement DESC) AS dense_rank_engagement
    FROM DuplicatedUserEngagement
),
EngagementWithDate AS (
    SELECT
        user_id,
        user_name,
        candidate,
        user_followers_count,
        total_tweets,
        total_engagement,
        follower_to_engagement_ratio,
        CURRENT_DATE AS query_date
    FROM ExtendedUserEngagement
),
RedundantAggregations AS (
    SELECT
        user_id,
        user_name,
        candidate,
        user_followers_count,
        SUM(total_tweets) AS redundant_total_tweets,
        SUM(total_engagement) AS redundant_total_engagement,
        AVG(follower_to_engagement_ratio) AS
redundant_follower_to_engagement_ratio
    FROM EngagementWithDate
    GROUP BY user_id, user_name, candidate,
user_followers_count
),
RepeatedEngagements AS (
    SELECT * FROM RedundantAggregations
    UNION ALL
    SELECT * FROM RedundantAggregations
    UNION ALL
    SELECT * FROM RedundantAggregations
),
UnnecessaryJoin AS (
    SELECT
        re.user_id,
        re.user_name,
        re.candidate,
        re.user_followers_count,
        re.redundant_total_tweets,
        re.redundant_total_engagement,
        re.redundant_follower_to_engagement_ratio,
        uw.query_date
    FROM RepeatedEngagements re
    LEFT JOIN EngagementWithDate uw ON re.user_id =
uw.user_id
),
ExcessiveFilters AS (
    SELECT *
    FROM UnnecessaryJoin
    WHERE user_followers_count > 1000
    AND candidate = 'Trump'
```

	`` AND user_name LIKE '%a%' AND user_id NOT IN (SELECT user_id FROM UnnecessaryJoin WHERE candidate = 'Biden')), DoubleSort AS (SELECT * FROM ExcessiveFilters ORDER BY redundant_follower_to_engagement_ratio DESC, user_id ASC), FinalSortAndLimit AS (SELECT * FROM DoubleSort ORDER BY candidate DESC, redundant_follower_to_engagement_ratio DESC, user_id ASC LIMIT 1000) SELECT user_id, user_name, candidate, user_followers_count, redundant_total_tweets AS total_tweets, redundant_total_engagement AS total_engagement, redundant_follower_to_engagement_ratio AS follower_to_engagement_ratio FROM FinalSortAndLimit ORDER BY candidate, redundant_follower_to_engagement_ratio DESC; ```	
Optimized	``` WITH UserEngagement AS (SELECT u.user_id, u.user_name, t.tweet_about AS candidate, u.user_followers_count, COUNT(t.tweet_id) AS total_tweets, SUM(t.likes + t.retweet_count) AS total_engagement, CASE WHEN u.user_followers_count = 0 THEN 0 ELSE ROUND(CAST(CAST(SUM(t.likes + t.retweet_count) AS FLOAT) / u.user_followers_count AS NUMERIC), 2) END AS followers_to_engagement_ratio FROM users u JOIN tweets t ON u.user_id = t.user_id GROUP BY u.user_id, u.user_name, t.tweet_about, u.user_followers_count) SELECT user_id, user_name, candidate, user_followers_count, total_tweets, total_engagement, followers_to_engagement_ratio FROM UserEngagement ```	1s 88ms

	``ORDER BY candidate, followers_to_engagement_ratio DESC;```	

Days With High Volume Of Tweets

Inefficient	```WITH DailyCandidateVolume AS (SELECT created_at::DATE AS tweet_date, tweet_about AS candidate, COUNT(tweet_id) AS tweet_count, SUM(likes + retweet_count) AS total_engagement FROM tweets WHERE tweet_about IN ('Biden', 'Trump') GROUP BY created_at::DATE, tweet_about), DuplicatedVolume AS (SELECT * FROM DailyCandidateVolume UNION ALL SELECT * FROM DailyCandidateVolume UNION ALL SELECT * FROM DailyCandidateVolume UNION ALL SELECT * FROM DailyCandidateVolume), HighVolumeDays AS (SELECT candidate, tweet_date, tweet_count, total_engagement, RANK() OVER(PARTITION BY candidate ORDER BY tweet_count DESC) AS daily_rank FROM DuplicatedVolume), FilteredHighVolumeDays AS (SELECT * FROM HighVolumeDays WHERE tweet_count > 0 -- Redundant filter), RepeatedVolumeFilter AS (SELECT * FROM FilteredHighVolumeDays UNION ALL SELECT * FROM FilteredHighVolumeDays), UnnecessaryJoin AS (SELECT vhd.candidate, vhd.tweet_date, vhd.tweet_count, vhd.total_engagement, vhd.daily_rank,```	>60s

	`` vhd.tweet_date AS extra_date FROM RepeatedVolumeFilter vhd LEFT JOIN tweets ts ON vhd.candidate = ts.tweet_about), RedundantRanking AS (SELECT candidate, tweet_date, tweet_count, total_engagement, daily_rank, ROW_NUMBER() OVER (PARTITION BY candidate ORDER BY total_engagement DESC) AS redundant_rank FROM UnnecessaryJoin), ExcessiveFilters AS (SELECT * FROM RedundantRanking WHERE redundant_rank <= 5 AND tweet_count > 0 -- Redundant filter again AND candidate IN ('Biden', 'Trump') -- Redundant filter), FinalSort AS (SELECT * FROM ExcessiveFilters ORDER BY candidate, tweet_date DESC), FinalOutput AS (SELECT tweet_date, candidate, tweet_count, total_engagement FROM FinalSort WHERE daily_rank <= 5 -- Filtering again for the same condition) SELECT tweet_date, candidate, tweet_count, total_engagement FROM FinalOutput ORDER BY candidate, tweet_date; ```	
Optimized	``` WITH DailyCandidateVolume AS (SELECT created_at::DATE AS tweet_date, tweet_about AS candidate, COUNT(tweet_id) AS tweet_count, SUM(likes + retweet_count) AS total_engagement FROM tweets GROUP BY created_at::DATE, tweet_about), HighVolumeDays AS (SELECT ```	432ms

```
      candidate,
      tweet_date,
      tweet_count,
      total_engagement,
      RANK() OVER(PARTITION BY candidate ORDER BY
tweet_count DESC) AS daily_rank
  FROM DailyCandidateVolume
)
SELECT
  tweet_date,
  candidate,
  tweet_count,
  total_engagement
FROM HighVolumeDays
WHERE daily_rank <= 5
ORDER BY candidate, tweet_date;
```