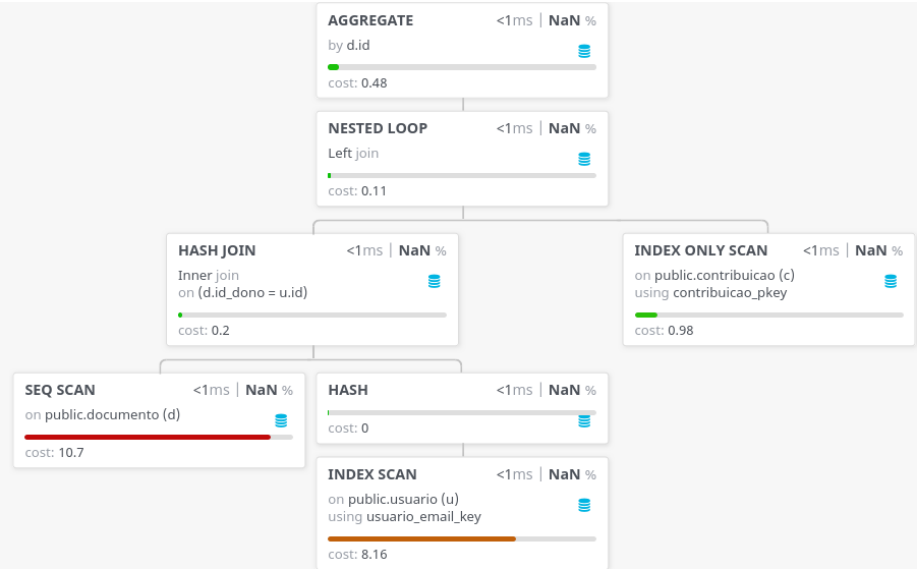
# Explicação de cada EXPLAIN

## Consulta 1

```
SELECT
    d.id,
    d.tipo,
    d.nome,
    d.descricao,
    COUNT(c.id_documento) AS numero_contribuidores
FROM documento d
INNER JOIN usuario u ON u.id = d.id_dono
LEFT JOIN contribuicao c ON d.id = c.id_documento
WHERE u.email = 'willian@email.com'
GROUP BY d.id;
```

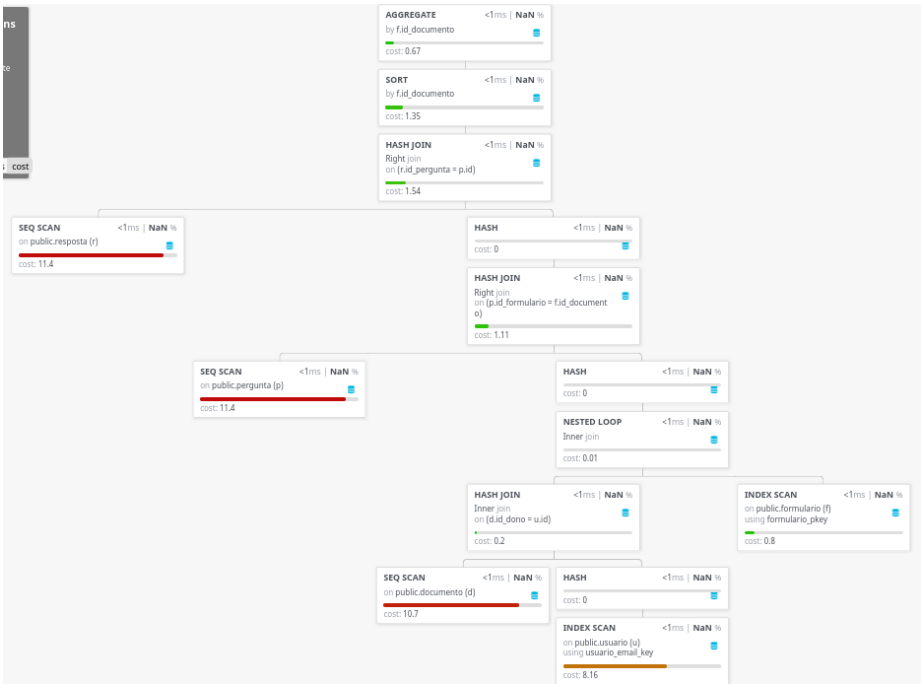



Os algoritmos do plano com maior custo são a busca sequencial e o índice sobre o campo de email. A busca sequencial por natureza tem um custo alto, pois todos os blocos precisam ser lidos. O índice sobre o campo de email, que é criado automaticamente pelo PostgreSQL por ser UNIQUE, tem um custo alto provavelmente porque o arquivo de dados não está organizado pelo campo email, então precisamos utilizar um índice secundário para encontrar os registros.

## Consulta 2

```
SELECT
    f.id_documento,
    f.data_limite,
```

```
    COUNT(DISTINCT r.id_usuario) as numero_respostas
FROM formulario f
INNER JOIN documento d ON f.id_documento = d.id
INNER JOIN usuario u ON d.id_dono = u.id
LEFT JOIN pergunta p ON f.id_documento = p.id_formulario
LEFT JOIN resposta r ON p.id = r.id_pergunta
WHERE u.email = 'luciano@email.com'Zx c
GROUP BY f.id_documento;
```

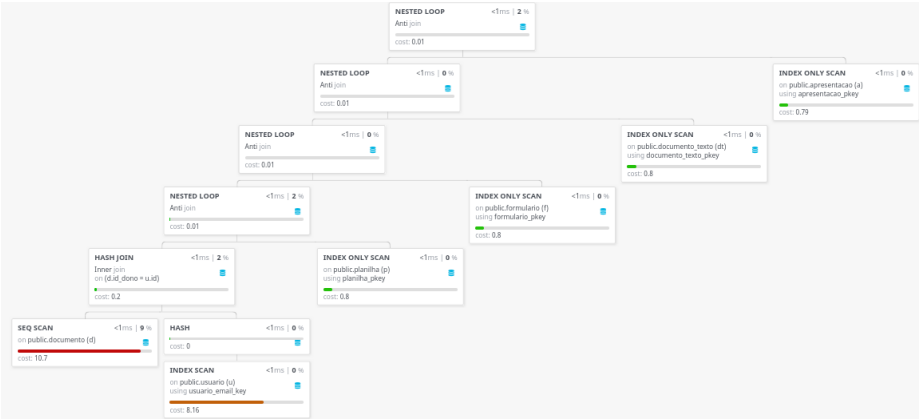



Nesse plano, precisamos de três buscas sequenciais, o que encarece muito o custo da consulta. Podemos criar índices para a tabela "resposta", pois ela naturalmente possui muitos registros, e ler sequencialmente todos os blocos do arquivo de dados deixa a consulta muito cara.

## Consulta 3

```
SELECT
    d.id,
    d.tipo,
    d.nome,
    d.descricao
FROM documento d
INNER JOIN usuario u ON d.id_dono = u.id
WHERE u.email = 'willian@email.com'
```

```
AND NOT EXISTS (
    SELECT 1
    FROM planilha p
    WHERE p.id_documento = d.id
) AND NOT EXISTS (
    SELECT 1
    FROM formulario f
    WHERE f.id_documento = d.id
) AND NOT EXISTS (
    SELECT 1
    FROM documento_texto dt
    WHERE dt.id_documento = d.id
) AND NOT EXISTS (
    SELECT 1
    FROM apresentacao a
    WHERE a.id_documento = d.id
);
```

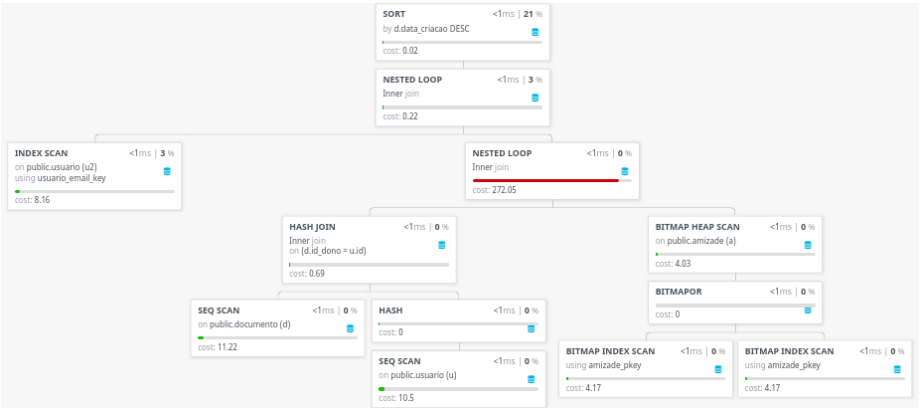



A tabela “documento” possui muito mais registros que as tabelas “planilha”, “formulario”, “documento_texto” e “apresentacao”, então o custo da busca sequencial é maior para essa tabela do que seria para as outras. Ainda assim, lida a tabela “documento”, provavelmente os registros ficarão em memória RAM, então o custo de ler sequencialmente a tabela “documento” é menor que o custo total da soma dos custos de ler sequencialmente as outras tabelas.

## Consulta 4

```
WITH amigos_usuario AS (
    SELECT
        u.id,
        u.nome
    FROM usuario u
```

```
    INNER JOIN amizade a ON u.id = a.id_remetente OR u.id = a.id_destinatario
    INNER JOIN usuario u2 ON
        (u.id = a.id_remetente AND u2.id = a.id_destinatario) OR
        (u.id = a.id_destinatario AND u2.id = a.id_remetente)
    WHERE u2.email = 'fatima@email.com'
)
SELECT
    d.id,
    d.nome,
    d.descricao,
    d.data_criacao,
    a.nome AS nome_amigo
FROM documento d
INNER JOIN amigos_usuario a ON d.id_dono = a.id
WHERE d.data_criacao >= NOW() - INTERVAL '1 week'
ORDER BY d.data_criacao DESC;
```




Percemos que o custo do loop aninhado é muito alto (dentre todas as outras consultas, é o que mais pesa). Além da falta de índices, o que pode estar causando esse problema é que estamos fundindo duas tabelas grandes, e como o algoritmo de loop aninhado custa (n x m) leituras de blocos (sendo n e m o número de registros de cada tabela), o custo acabou ficando muito alto.