

# Relatório

Antônio Romano

Caio Paranhos

Davi Santana

Igor Monárdez

2 de julho de 2024

## 1 Visões

Foram criadas quatro visões: três visões virtuais e uma materializada. As visões virtuais não armazenam dados fisicamente. Cada vez que uma visão virtual é consultada, o banco de dados executa a consulta subjacente e retorna os resultados. As visões virtuais criadas foram: `canais_patrocinados`, `user_subscriptions` e `video_doacoes`.

A visão materializada criada foi a `top_canais`. Diferente das visões virtuais, uma visão materializada armazena seus resultados fisicamente, como uma tabela. Essas visões são úteis quando há consultas que demoram muito para serem executadas, pois os resultados são armazenados e não precisam ser recalculados a cada consulta. No entanto, as visões materializadas precisam ser atualizadas manualmente ou automaticamente em intervalos regulares para refletir as alterações nos dados subjacentes.

### 1.1 Canais Patrocinados

Esta visão mostra os canais que são patrocinados e os valores de patrocínio por empresa. Ela junta as tabelas `Patrocinio` e `Empresa` para obter essas informações. Essa visão foi utilizada para auxiliar na primeira consulta: Identificar quais são os canais patrocinados e os valores de patrocínio pagos por empresa.

```
CREATE OR REPLACE VIEW canais_patrocinados AS
SELECT
    patr.nome_canal,
    empr.nome AS empresa_pagadora,
    patr.valor AS valor_patrocinio
FROM
    Patrocinio patr
INNER JOIN
    Empresa empr ON empr.nro = patr.nro_empresa;
```

Figura 1: Visão virtual canais\_patrocinados

## 1.2 User Subscriptions

Esta visão mostra a quantidade de canais que cada usuário é membro e o valor total que eles desembolsaram. Ela junta as tabelas inscricao e nivelcanal para obter essas informações. Essa visão foi utilizada para auxiliar na segunda consulta: Descobrir de quantos canais cada usuário é membro e a soma do valor desembolsado por usuário por mês.

```
CREATE OR REPLACE VIEW user_subscriptions AS
SELECT
    i.nick_membro AS usuario,
    COUNT(i.nome_canal) AS total_canais,
    SUM(nc.valor) AS total_gasto
FROM
    trabbd2.inscricao i
JOIN
    trabbd2.nivelcanal nc
ON
    i.nome_canal = nc.nome_canal AND i.nivel = nc.nivel
GROUP BY
    i.nick_membro;
```

Figura 2: Visão virtual user\_subscriptions

### 1.3 Video Doações

Esta visão lista os vídeos e a soma dos valores de doações recebidas. Ela junta as tabelas doacao, video e comentario para obter essas informações. Essa visão foi utilizada para auxiliar na quarta consulta: Identificar quais são os canais patrocinados e os valores de patrocínio pagos por empresa.

```
CREATE OR REPLACE VIEW video_doacoes AS
SELECT
    v.id_video AS id_video,
    SUM(d.valor) AS total_doacoes
FROM
    trabbd2.doacao d
JOIN
    trabbd2.video v ON d.id_video = v.id_video
JOIN
    trabbd2.comentario c ON d.id_video = c.id_video AND d.nick_usuario = c.nick_usuario AND d.seq_comentario = c.seq
WHERE c.coment_on = TRUE
GROUP BY
    v.id_video
ORDER BY
    total_doacoes DESC;
```

Figura 3: Visão virtual video\_doacoes

### 1.4 Top Canais

Esta visão materializada mostra os maiores valores por canal. Ela junta as tabelas canal, patrocínio, inscricao, nivelcanal e doacao para obter essas informações. Essa visão foi utilizada para auxiliar em mais de uma consulta, são elas:

- Consulta 3 - Listar e ordenar os canais que já receberam doações e a soma dos valores recebidos em doação.
- Consulta 5 - Listar e ordenar os k canais que mais recebem patrocínio e os valores recebidos.
- Consulta 6 - Listar e ordenar os k canais que mais recebem aportes de membros e os valores recebidos.

- Consulta 7 - Listar e ordenar os k canais que mais receberam doações considerando todos os vídeos.
- Consulta 8 - Listar os k canais que mais faturam considerando as três fontes de receita: patrocínio, membros inscritos e doações.

```

1 CREATE MATERIALIZED VIEW top_canais AS
2 SELECT
3   c.nome AS nome_canal,
4   p.qtd_patrocinio,
5   COALESCE(SUM(p.total_patrocinio), 0) AS total_patrocinio,
6   m.qtd_aportes,
7   COALESCE(SUM(m.total_aportes), 0) AS total_aportes,
8   d.qtd_doacoes,
9   COALESCE(SUM(d.total_doacoes), 0) AS total_doacoes,
10  total_patrocinio + total_aportes + total_doacoes AS total_faturamento
11 FROM
12   trabbd2.canal c
13 LEFT JOIN (
14   SELECT
15     p.nome_canal,
16     COUNT(p.valor) AS qtd_patrocinio,
17     SUM(p.valor) AS total_patrocinio
18   FROM
19     trabbd2.patrocinio p
20   GROUP BY
21     p.nome_canal
22   ) p ON c.nome = p.nome_canal
23 LEFT JOIN (
24   SELECT
25     nc.nome_canal,
26     COUNT(nc.valor) AS qtd_aportes,
27     SUM(nc.valor) AS total_aportes
28   FROM
29     trabbd2.inscricao i
30   JOIN
31     trabbd2.nivelcanal nc ON i.nome_canal = nc.nome_canal AND i.nivel = nc.nivel
32   GROUP BY
33     nc.nome_canal
34   ) m ON c.nome = m.nome_canal
35 LEFT JOIN (
36   SELECT
37     v.nome_canal,
38     COUNT(d.valor) AS qtd_doacoes,
39     SUM(d.valor) AS total_doacoes
40   FROM
41     trabbd2.doacao d
42   JOIN
43     trabbd2.video v ON d.id_video = v.id_video
44   GROUP BY
45     v.nome_canal
46   ) d ON c.nome = d.nome_canal
47 GROUP BY
48   c.nome, p.qtd_patrocinio, c.nome, m.qtd_aportes, d.qtd_doacoes, total_patrocinio +
  total_aportes + total_doacoes;

```

Figura 4: Visão materializada top\_canais

## 2 Índice de Apoio

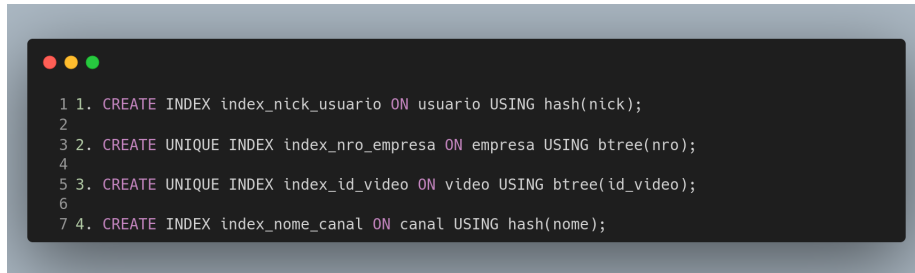


Figura 5: Criação dos índices de apoio

### 2.1 Index Nick Usuario

Para otimizar as operações de busca na tabela usuário, foi criado um índice hash sobre o campo nick, que é a chave primária natural composta por uma string. Dado que esta chave primária possui uma cláusula única, o índice hash não introduz um overhead significativo, garantindo que a complexidade de busca se aproxime de  $O(1)$ . A implementação deste índice hash é particularmente vantajosa para o escopo do projeto final desta disciplina, facilitando buscas rápidas e eficientes por comentários e inscrições associados aos usuários.

### 2.2 Index Nome Canal

Para otimizar as operações de busca na tabela canal, foi criado um índice hash sobre o campo nome, que é a chave primária natural composta por uma string. Considerando que esta chave primária possui uma cláusula única, o índice hash não introduz um overhead significativo, garantindo que a complexidade de busca se aproxime de  $O(1)$ . Este índice é especialmente vantajoso para o escopo do projeto final, pois facilita as buscas por aportes, doações e patrocínios, que correspondem às procedures 3, 5, 6, 7 e 8. Além disso, contribui significativamente para a criação das views e triggers associadas.

### 2.3 Index Id Video

Foi criado um índice B-tree sobre o campo id\_video, uma chave primária artificial no formato integer, que é ideal para esse tipo de índice. Este índice

aumentará significativamente a velocidade de busca ao consultar as doações por vídeo, que está dentro do escopo da procedure 4.

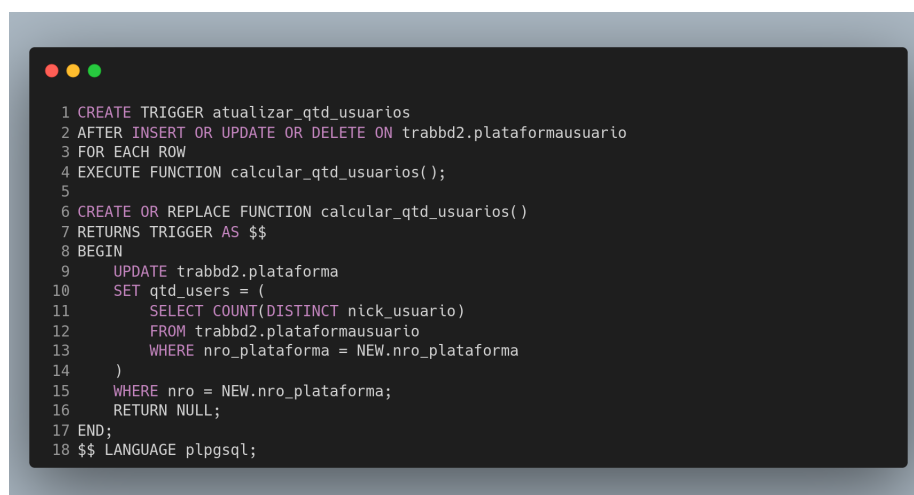
## 2.4 Index Número Empresa

Foi criado um índice B-tree sobre o campo nro da tabela empresa, que é uma chave primária artificial formada por um integer. Este índice é utilizado ao buscar as doações feitas aos canais por cada empresa, conforme a view da procedure 1. Consequentemente, aumentará a eficiência de busca nessa procedure.

## 3 Triggers

Foram criados dois triggers para manter a consistência do banco de dados no momento da estruturação dos dados:

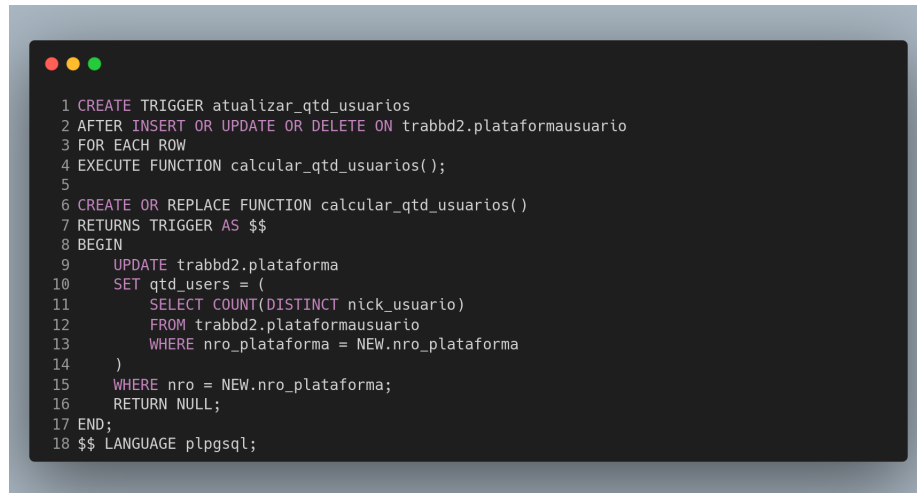
- `atualizar_qtd_usuarios`: Este trigger é acionado após a inserção, atualização ou exclusão de um registro na tabela `plataformausuario`. Para cada uma dessas ações, ele executa a função `calcular_qtd_usuarios()`. Essa função atualiza a quantidade de usuários na tabela `plataforma` com base na quantidade de usuários únicos na tabela `plataformausuario` para a plataforma correspondente.



```
1 CREATE TRIGGER atualizar_qtd_usuarios
2 AFTER INSERT OR UPDATE OR DELETE ON trabbd2.plataformausuario
3 FOR EACH ROW
4 EXECUTE FUNCTION calcular_qtd_usuarios();
5
6 CREATE OR REPLACE FUNCTION calcular_qtd_usuarios()
7 RETURNS TRIGGER AS $$
8 BEGIN
9     UPDATE trabbd2.plataforma
10    SET qtd_users = (
11        SELECT COUNT(DISTINCT nick_usuario)
12        FROM trabbd2.plataformausuario
13        WHERE nro_plataforma = NEW.nro_plataforma
14    )
15    WHERE nro = NEW.nro_plataforma;
16    RETURN NULL;
17 END;
18 $$ LANGUAGE plpgsql;
```

Figura 6: Trigger para atualizar a quantidade de usuários em uma plataforma

- `atualizar_qtd_visualizacoes`: Este trigger é acionado após a inserção, atualização ou exclusão de um registro na tabela `Video`. Para cada uma dessas ações, ele executa a função `calcular_qtd_visualizacoes()`. Essa função atualiza a quantidade de visualizações na tabela `Canal` com base na soma das visualizações totais dos vídeos na tabela `Video` para o canal correspondente.



```

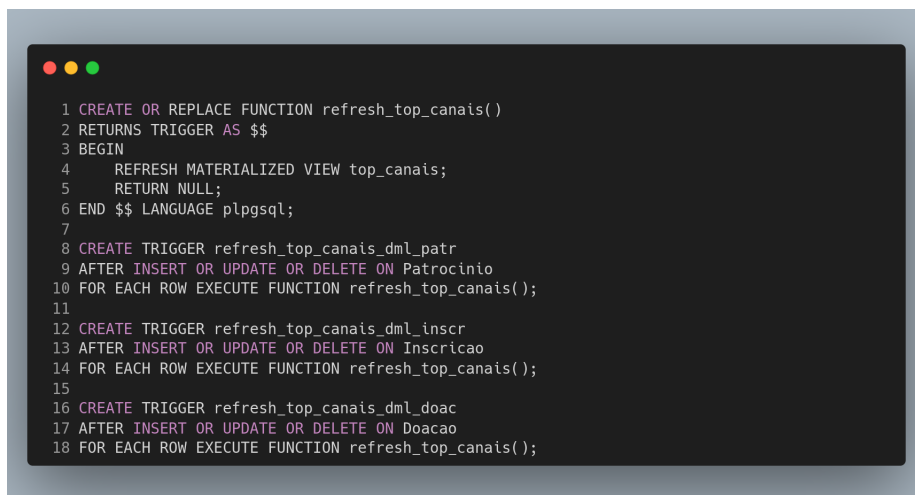
1 CREATE TRIGGER atualizar_qtd_usuarios
2 AFTER INSERT OR UPDATE OR DELETE ON trabbd2.plataformausuario
3 FOR EACH ROW
4 EXECUTE FUNCTION calcular_qtd_usuarios();
5
6 CREATE OR REPLACE FUNCTION calcular_qtd_usuarios()
7 RETURNS TRIGGER AS $$
8 BEGIN
9     UPDATE trabbd2.plataforma
10    SET qtd_users = (
11        SELECT COUNT(DISTINCT nick_usuario)
12        FROM trabbd2.plataformausuario
13        WHERE nro_plataforma = NEW.nro_plataforma
14    )
15    WHERE nro = NEW.nro_plataforma;
16    RETURN NULL;
17 END;
18 $$ LANGUAGE plpgsql;

```

Figura 7: Trigger para atualizar a quantidade de visualizações de um vídeo

Além disso, para garantir a consistência dos dados para a view materializada `top_canais`, foi criada também a função `refresh_top_canais`. Essa função é responsável por atualizar a view `top_canais` e ela é chamada a partir dessas triggers:

- `refresh_top_canais_dml_patr`: Trigger na tabela `Patrocinio` que é acionada sempre que um registro é inserido, atualizado ou excluído nesta tabela.
- `refresh_top_canais_dml_inscr`: Trigger na tabela `Inscrição` que é acionada sempre que um registro é inserido, atualizado ou excluído nesta tabela.
- `refresh_top_canais_dml_doac`: Trigger na tabela `Doação` que é acionada sempre que um registro é inserido, atualizado ou excluído nesta tabela.



```

1 CREATE OR REPLACE FUNCTION refresh_top_canais()
2 RETURNS TRIGGER AS $$
3 BEGIN
4     REFRESH MATERIALIZED VIEW top_canais;
5     RETURN NULL;
6 END $$ LANGUAGE plpgsql;
7
8 CREATE TRIGGER refresh_top_canais_dml_patr
9 AFTER INSERT OR UPDATE OR DELETE ON Patrocinio
10 FOR EACH ROW EXECUTE FUNCTION refresh_top_canais();
11
12 CREATE TRIGGER refresh_top_canais_dml_inscr
13 AFTER INSERT OR UPDATE OR DELETE ON Inscricao
14 FOR EACH ROW EXECUTE FUNCTION refresh_top_canais();
15
16 CREATE TRIGGER refresh_top_canais_dml_doac
17 AFTER INSERT OR UPDATE OR DELETE ON Doacao
18 FOR EACH ROW EXECUTE FUNCTION refresh_top_canais();

```

Figura 8: Triggers para view materializada

Criar/Atualizar uma view materializada pode ser um processo custoso em termos de desempenho, pois exige a reexecução da consulta subjacente e a reescrita dos resultados na view materializada. No caso, a view *top/anaiséumaconsulta que pode ser be*

Embora possamos utilizar uma view virtual, as views materializadas são particularmente úteis para consultas pesadas e demoradas, pois armazenam o resultado da consulta e apenas o atualizam quando explicitamente solicitado. Isso pode economizar tempo e recursos do sistema, especialmente se a consulta envolver operações complexas, como junções de várias tabelas, agregações ou funções de janela.

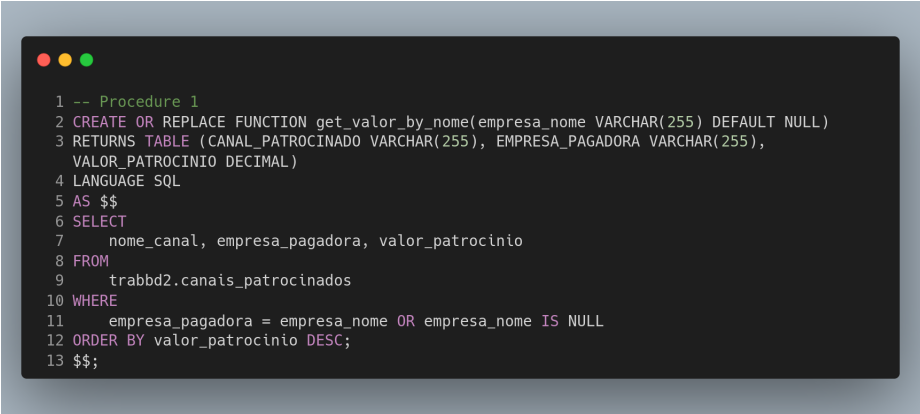
No entanto, é crucial lembrar que as views materializadas não são atualizadas automaticamente quando os dados subjacentes mudam. Portanto, é necessário encontrar um equilíbrio entre a frequência de atualização da view materializada e o desempenho da consulta. Se os dados subjacentes mudarem com frequência e você precisar de resultados atualizados em tempo real, uma view materializada pode não ser a melhor opção.

Triggers para atualizar a view materializada após eventos de inserção, atualização ou exclusão nas tabelas Patrocinio, Inscricao e Doacao foram criadas. Com a quantidade de dados que temos, essa é uma operação simples e pouco custosa, tornando válida a utilização das triggers. No entanto, caso esse processo se torne custoso com o tempo, pode-se alterar a estratégia de atualização da view para uma cronjob, atualizando a view com uma frequência pré-determinada.



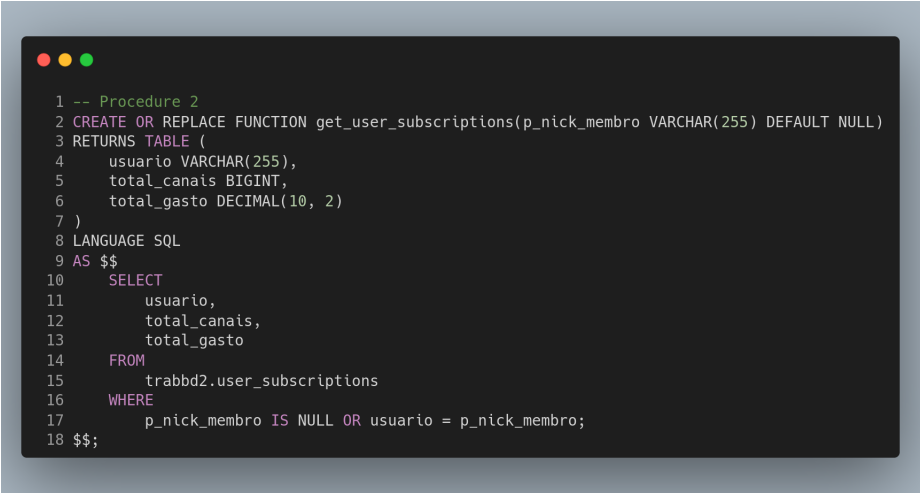
## 4 Procedures

As stored procedures foram escritas em SQL. Elas respondem as 8 consultas solicitadas e utilizam as views que foram criadas anteriormente.



```
1 -- Procedure 1
2 CREATE OR REPLACE FUNCTION get_valor_by_nome(empresa_nome VARCHAR(255) DEFAULT NULL)
3 RETURNS TABLE (CANAL_PATROCINADO VARCHAR(255), EMPRESA_PAGADORA VARCHAR(255),
4 VALOR_PATROCINIO DECIMAL)
5 LANGUAGE SQL
6 AS $$
7 SELECT
8     nome_canal, empresa_pagadora, valor_patrocinio
9 FROM
10     trabbd2.canais_patrocinados
11 WHERE
12     empresa_pagadora = empresa_nome OR empresa_nome IS NULL
13 ORDER BY valor_patrocinio DESC;
14 $$;
```

Figura : Procedure 1



```
1 -- Procedure 2
2 CREATE OR REPLACE FUNCTION get_user_subscriptions(p_nick_membro VARCHAR(255) DEFAULT NULL)
3 RETURNS TABLE (
4     usuario VARCHAR(255),
5     total_canais BIGINT,
6     total_gasto DECIMAL(10, 2)
7 )
8 LANGUAGE SQL
9 AS $$
10 SELECT
11     usuario,
12     total_canais,
13     total_gasto
14 FROM
15     trabbd2.user_subscriptions
16 WHERE
17     p_nick_membro IS NULL OR usuario = p_nick_membro;
18 $$;
```

Figura : Procedure 2

```

1 -- Procedure 3
2 CREATE OR REPLACE FUNCTION get_canal_doacoes(p_nome_canal VARCHAR(255) DEFAULT NULL)
3 RETURNS TABLE (
4     nome_canal VARCHAR(255),
5     total_doacoes DECIMAL(10, 2)
6 )
7 LANGUAGE SQL
8 AS $$
9     SELECT
10         nome_canal,
11         total_doacoes
12     FROM
13         trabbd2.top_canais
14     WHERE
15         p_nome_canal IS NULL OR nome_canal = p_nome_canal
16     ORDER BY
17         total_doacoes DESC;
18 $$;

```

Figura : Procedure 3

```

1 -- Procedure 4
2 CREATE OR REPLACE FUNCTION get_video_doacoes(p_id_video INT DEFAULT NULL)
3 RETURNS TABLE (
4     id_video INT,
5     total_doacoes DECIMAL(10, 2)
6 )
7 LANGUAGE SQL
8 AS $$
9     SELECT
10         id_video,
11         total_doacoes
12     FROM
13         trabbd2.video_doacoes
14     WHERE
15         (p_id_video IS NULL OR id_video = p_id_video)
16     ORDER BY
17         total_doacoes DESC;
18 $$;

```

Figura : Procedure 4

```

1 -- Procedure 5
2 CREATE OR REPLACE FUNCTION get_top_k_canais_patrocinio(k INT)
3 RETURNS TABLE (
4     nome_canal VARCHAR(255),
5     qtd_patrocinio DECIMAL(10,2),
6     total_patrocinio DECIMAL(10, 2)
7 )
8 LANGUAGE sql
9 AS $$
10 SELECT
11     nome_canal,
12     qtd_patrocinio,
13     total_patrocinio
14 FROM
15     trabbd2.top_canais
16 ORDER BY
17     qtd_patrocinio DESC
18 LIMIT k;
19 $$;

```

Figura : Procedure 5

```

1 -- Procedure 6
2 CREATE OR REPLACE FUNCTION get_top_k_canais_aportes(k INT)
3 RETURNS TABLE (
4     nome_canal VARCHAR(255),
5     qtd_aportes DECIMAL(10,2),
6     total_aportes DECIMAL(10, 2)
7 )
8 LANGUAGE SQL
9 AS $$
10 SELECT
11     nome_canal,
12     qtd_aportes,
13     total_aportes
14 FROM
15     trabbd2.top_canais
16 ORDER BY
17     qtd_aportes DESC
18 LIMIT k;
19 $$;

```

Figura : Procedure 6

```

1 -- Procedure 7
2 CREATE OR REPLACE FUNCTION get_top_k_canais_doacoes(k INT)
3 RETURNS TABLE (
4     nome_canal VARCHAR(255),
5     qtd_doacoes DECIMAL(10, 2)
6 )
7 LANGUAGE SQL
8 AS $$
9     SELECT
10         nome_canal,
11         qtd_doacoes
12     FROM
13         trabbd2.top_canais
14     ORDER BY
15         qtd_doacoes DESC
16     LIMIT k;
17 $$;

```

Figura : Procedure 7

```

1 -- Procedure 8
2 CREATE OR REPLACE FUNCTION get_top_k_canais_faturamento(k INT)
3 RETURNS TABLE (
4     nome_canal VARCHAR(255),
5     total_faturamento DECIMAL(10, 2)
6 )
7 LANGUAGE sql
8 AS $$
9     SELECT
10         nome_canal,
11         total_faturamento
12     FROM
13         trabbd2.top_canais
14     ORDER BY
15         total_faturamento DESC
16     LIMIT k;
17 $$;

```

Figura : Procedure 8