UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS E TECNOLOGIAS PARA A SUSTENTABILIDADE

CAMPUS SOROCABA

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LABORATÓRIO DE BANCO DE DADOS E ENGENHARIA DE SOFTWARE 2

PROJETO INTEGRADO

GRUPO 8 - PROGRAMA + NATUREZA DA DESPESA

DOCENTES: ALEXANDRE ÁLVARO

SAHUDY MONTENEGRO GONZÁLEZ

ALUNOS: HENRIQUE EIHARA: 490016

GABRIELA DE JESUS MARTINS: 489689

GUSTAVO RODRIGUES: 489999

VALDEIR SOARES PEROZIM: 489786

Data de entrega: 08/06/2015

Entrega Final

Sorocaba

2015

Sumário

1 Especificação das Consultas

1.1 Total de Gastos por Natureza da Despesa

A consulta apresentada na Listagem 1 realiza a soma de todas as despesas agrupadas em uma determinada categoria de despesa (atributo `código` da tabela de **natureza**).

O atributo `descricao` do município pode ser informado, permitindo ao usuário especificar qual município deseja consultar. Porém, a consulta não é relativa, apenas nomes exatos retornarão resultado, assim evitaremos que dados de despesas de diferentes municípios misturem-se atrapalhando a visualização dos resultados.

A consulta retorna os seguintes atributos: `codigo`, `descricao` e `gasto`, que é uma coluna representando o resultado da agregação renomeada desta forma e está em ordem decrescente.

```
1 SELECT N.codigo, N.descricao, SUM(Desp.valor) AS gasto
2 FROM despesa Desp, natureza N, (SELECT M.codigo FROM municipio M
     WHERE M.descricao = <cidade>)Mun
3 WHERE Mun.codigo = Desp.codigomunicipio AND N.codigo = Desp.
     codigonatureza
4 GROUP BY N.codigo, N.descricao
5 ORDER BY gasto DESC;
```

Listing 1: Total de gastos por natureza de despesa

1.2 Total de Gastos por Programa

Na consulta da Listagem 2 também será realizada uma agregação totalizando os valores das despesas, porém, agrupadas por programa de determinado município, representado, respectivamente, pelos atributos `codigo` e `descricaointernamunicipio` da tabela **programa**.

O retorno da consulta contém os seguites campos: `codigo`, `descricaointernamunicipio` e `gasto`, que é o resultado da agregação. O resultado é ordenado decrescentemente pelo atributo `gasto`.

```
1 SELECT P.codigo, P.descricaointernamunicipio, SUM(Desp.valor) AS
    gasto
2 FROM despesa Desp, programa P, (SELECT M.codigo FROM municipio M
3 WHERE M.descricao = <cidade>)Mun WHERE
4 Mun.codigo = Desp.codigomunicipio AND P.codigo = Desp.codigoprograma
5 GROUP BY P.codigo, P.descricaointernamunicipio
6 ORDER BY gasto DESC;
```

Listing 2: Total de gastos por programa

1.3 Gastos por Natureza de Despesa para cada Programa

A consulta da 3 recebe como parâmetros a descrição de duas naturezas de gasto para filtrar o atributo `descricao` da tabela **natureza**, um período de tempo que deseja-se analisar através do atributo `dataano` e uma faixa de valores para filtrar o atributo `valor` da tabela **despesa**.

Como resultado é retornada uma lista agrupando os valores de despesa totais para todas as categorias de natureza(atributo `descricao` de todos os programas(atributo `descricaointernamunicipio`) do município. Os resultados são filtrados por faixa de valores e por período de tempo(ano inicial e ano final) e ordenados pela descrição interna do programa no município.

```
1 SELECT P.descricaointernamunicipio, N.descricao, SUM(D.valor) AS
    gasto
2 FROM despesa D, programa P, (SELECT codigo, descricao FROM natureza
    WHERE (descricao ILIKE <natureza_1> OR descricao ILIKE <
    natureza_2>)) N
3 WHERE D.codigoprograma = P.codigo AND D.codigonatureza = N.codigo
    AND (dataano >= <data_ini> AND dataano <= <data_fim>)
4 GROUP BY P.descricaointernamunicipio, N.descricao
5 HAVING SUM(D.valor) > <limite_inf> AND SUM(D.valor) <= <limite_sup>
6 ORDER BY P.descricaointernamunicipio
```

Listing 3: Programa X Natureza Despesa

2 Técnicas de Acesso Eficiente ao Banco de Dados

Trabalhamos com as relações originais do banco, isto é, não alteramos qualquer tabela ou desnormalizamos o banco de dados.

Apesar da tabela **despesa** apresentar campos redundantes com as tabelas de **natureza** e **programa**, utilizamos apenas os campos *codigonatureza* e *codigoprograma*, e a partir deles recuperamos as informações necessárias nas tabelas originais, evitando problemas com dados inconsistentes.

2.1 Índices

A Listagem 4 apresenta os índices *hash* criados para os campos de chave estrangeira *codigonatureza* e *codigoprograma* da tabela **despesa**. Como nossas consultas realizam um produto cartesiano em que se compara estes campos, as consultas tiveram seus tempos de execução significativamente melhorados quando informamos um descrição específica de natureza ou de programa.

```
1 CREATE INDEX indice_codNatureza ON despesa USING hash (
    codigonatureza);
2 CREATE INDEX indice_codPrograma ON despesa USING hash (
    codigoprograma);
3 CREATE INDEX indice_dataano ON despesa USING btree(dataano);
```

Listing 4: Código para criação dos índices

2.2 Consulta Auxiliar no campo `dataano`

Criamos também um índice para o campo `dataano`, pois otimizou uma consulta auxiliar utilizada para tratar valores não informados à *stored procedure* da Listagem 7.

Com esta otimização, houve um ganho de 99,4% para esta consulta, que pode ser observado nos planos de consulta da Figura 1 que mostra a consulta sem índice. Neste caso é feita uma varredura sequencial em todas as tuplas da tabela **despesa**. Porém, a Figura 2 mostra que o otimizador conseguiu utilizar o índice criado para consulta realizada quando uma natureza específica é informada.

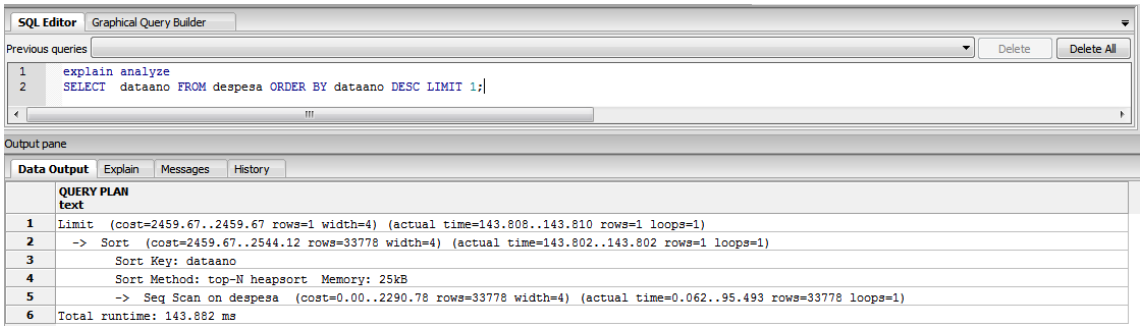

Figura 1: Consulta em `dataano` sem índice

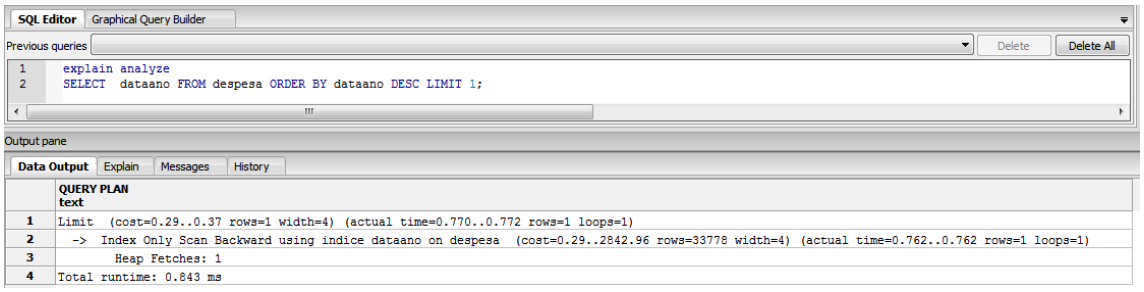

Figura 2: Consulta em `dataano` usando índice

2.3 Consulta Total de Gastos por Natureza

A versão da Consulta 1 que realiza a agregação de todos os possíveis valores quando uma descrição específica de natureza não é informada, não foi afetada por este índice, pois o BD precisa ler todas as tuplas para realizar as totalizações como pode ser visto na Figura 15.

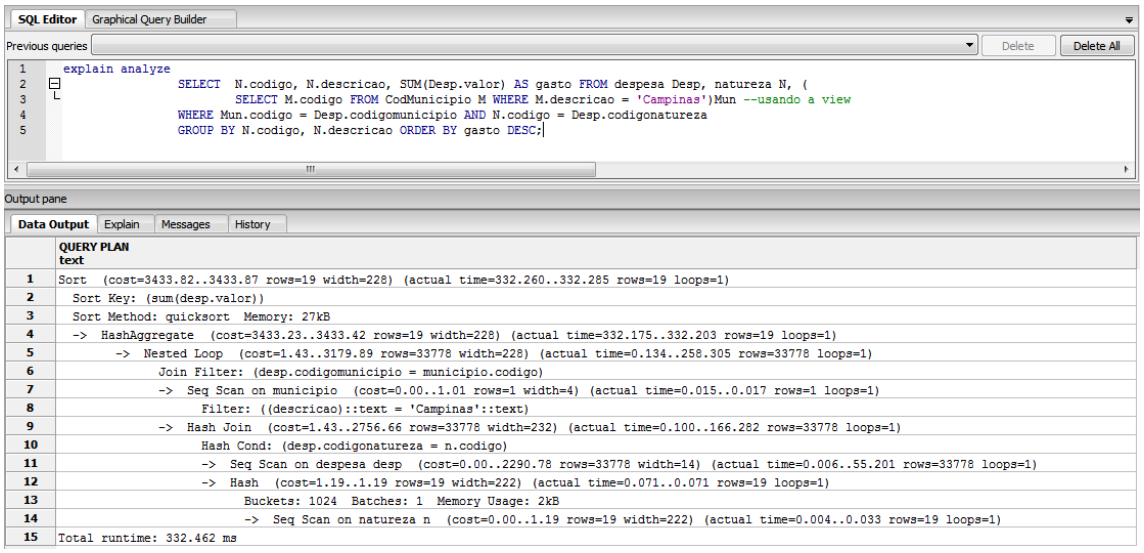

Figura 3: Consulta Simples Natureza Agregando Todos os Valores

Quando o usuário informa o parâmetro `descricao` para natureza, então o índice no campo `codigodescricao` em **despesa** é utilizado para otimizar a consulta como mostra a linha 15 do plano de consulta da Figura 4.

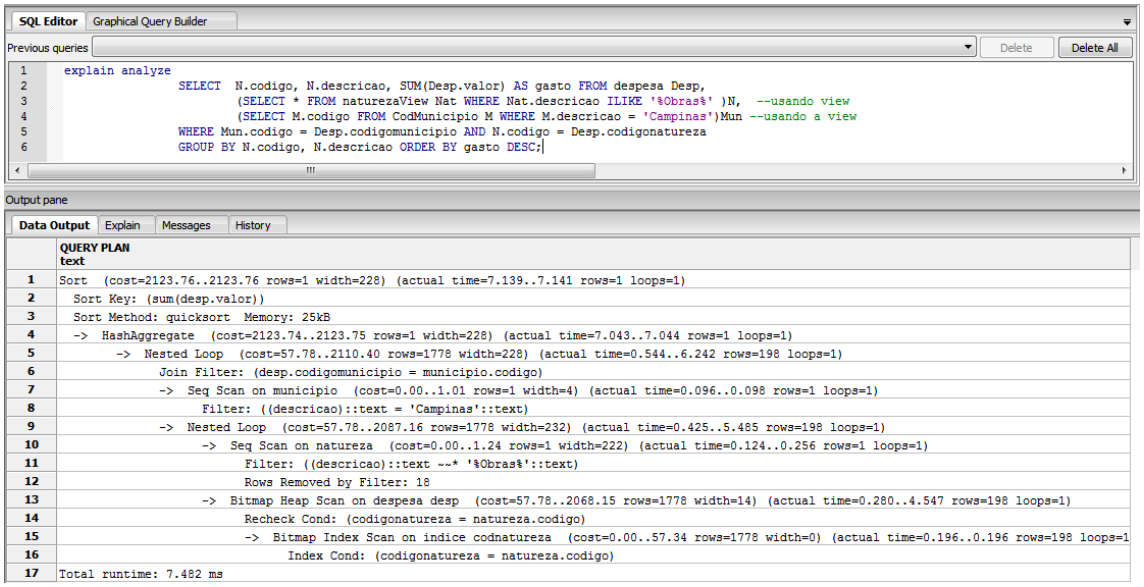

Figura 4: Consulta Simples Natureza Específica com Índice

2.4 Consulta Total de Gastos por Programa

O mesmo ocorreu com a Consulta 2 quando uma descrição específica para o programa é fornecida. As linhas 15 e 16 da Figura 5 mostra a utilização do índice criado para o campo `codigoprograma`.

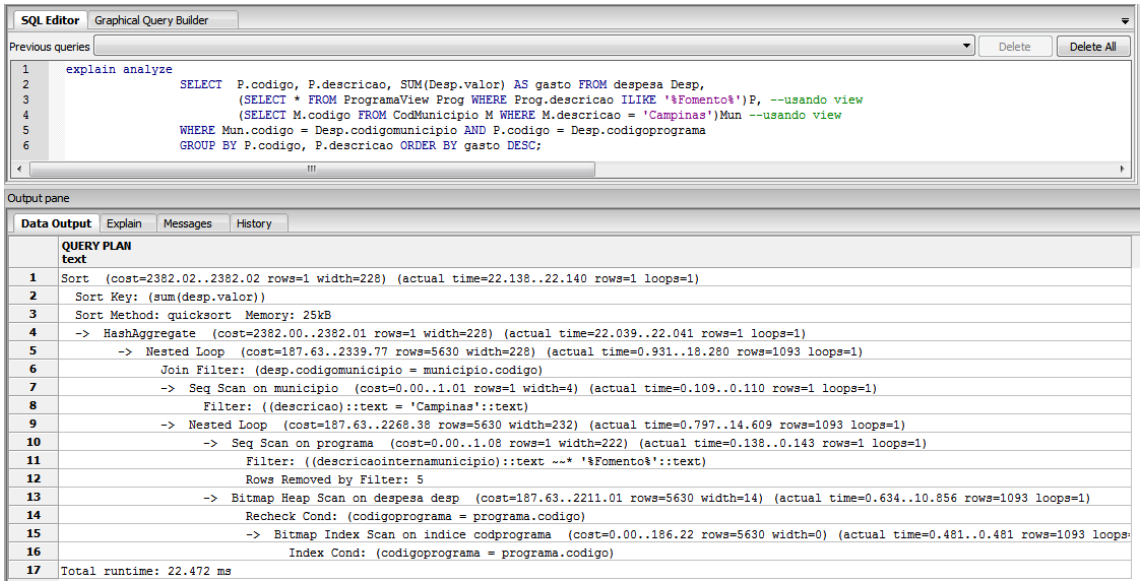

Figura 5: Consulta Simples Programa Específica com Índice

A redução no tempo de execução da consulta foi de aproximadamente 85% como podemos observar na Figura 6, quando a consulta é realizada sem a presença de índice.

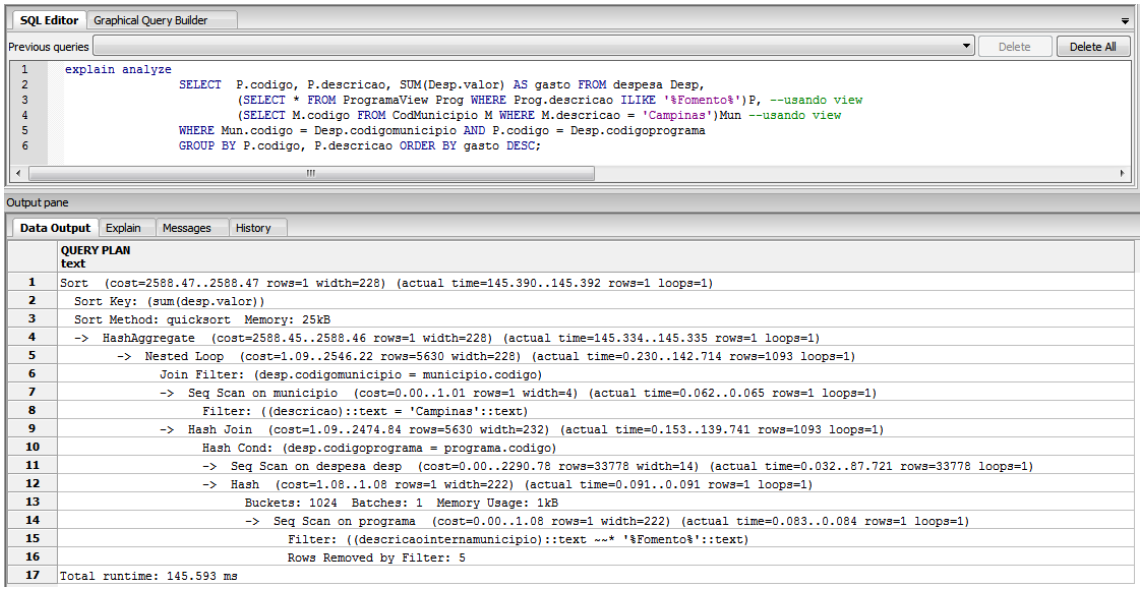

Figura 6: Consulta Simples Programa Específica sem Índice

2.5 Consulta Avançada

Para esta consulta, os índices `indice_CodNatureza` e `indice_CodPrograma` não foram utilizados pelo otimizador.

No caso do índice `indice_CodPrograma`, para agregar os valores de todas as despesas de certo programa é necessário ler todas as entradas de todos os códigos de programas na tabela **despesa**, o que dispensa a utilização de índices, pois é realizada uma leitura sequencial de todas as tuplas como mostra a Figura 7.

Não aplicamos índice no campo descrição de natureza, por exemplo, pois utilizamos consulta com ILIKE e curingas no padrão '%Fomento%' que impossibilita a utilização de índices, pois é necessário aplicar o padrão à todas as tuplas ou em todo o índice.

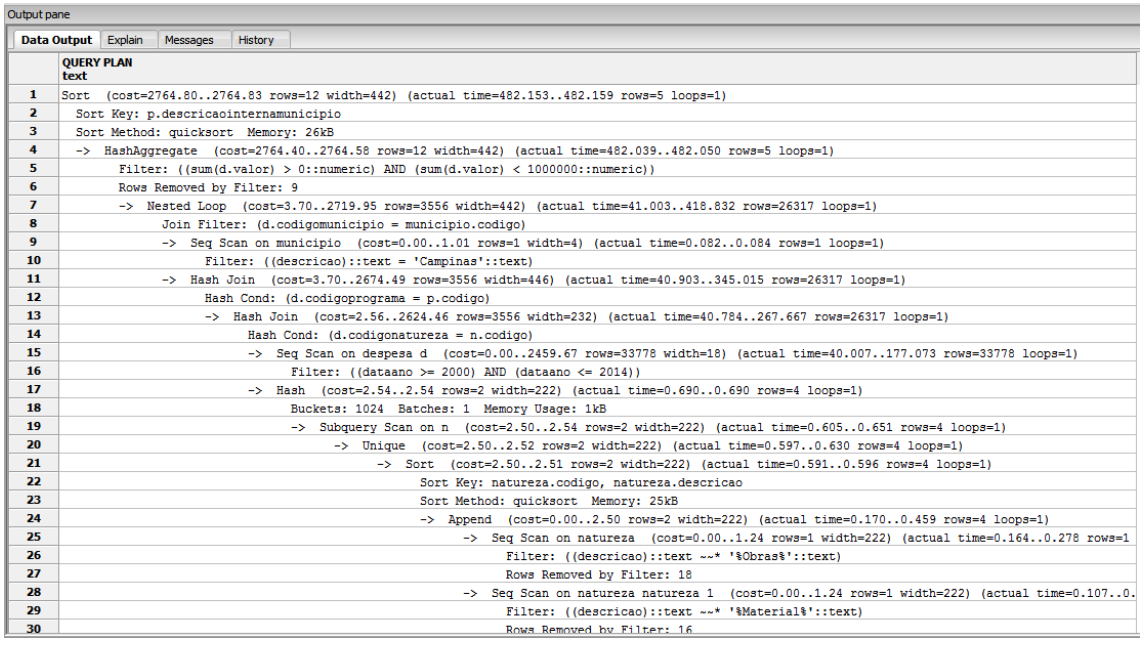

Figura 7: Plano de Consulta para Consulta Avançada

2.6 Tempos de Consulta

A Tabela 1 foi criada a partir do tempo de consulta medido a partir de um *SELECT* na *stored procedure* associada à consulta.

Não utilizamos as stored procedures para analisar o plano de consulta e utilização de índices, pois o comando `explain analyze` não consegue exibir todos os passos executados para realizar a consulta quando se trata de *stored procedures*, apenas seu tempo de execução.

Em algumas consultas não consideramos a diferença de desempenho com e sem índice, pois esta diferença não se deve à utilização de índices pelo otimizador como explicamos anteriormente analisando os planos de consulta. Os tempos estão nas mesmas ordens de grandeza e a diferença provavelmente se deve a diferentes disponibilidades de processamento e memória no momento da medição.

A metodologia que adotamos para realizar as medições foi a seguinte: Todas as medições foram realizadas em uma mesma máquina rodando postgres SQL 9.3 no sistema operacional Windows 7 Home Edition. O serviço do postgres foi reinicializado para cada consulta.

Para os valores sem índice, utilizamos o primeiro resultado, que em média representa o pior desempenho da consulta. E para a consulta com índice o segundo resultado da medição, que apresentou o tempo nos melhores patamares. Assim, foi possível identificar o maior ganho possível com a utilização de índices.

Consultas	Sem Índice(ms)	Com Índice(ms)	Redução
SELECT dataano	143,882	0.843	99,4%
Consulta 1, natureza específica	88,384	2,723	97%
Consulta 1, todas as naturezas	132,177	129,322	-
Consulta 2, programa específico	128,718	5,470	96%
Consulta 2, todos os programas	123,313	151,148	-
Consulta avançada, com todos os parâmetros	158,533	176,248	-

Tabela 1: Tempos de Consulta

3 Programação com Banco de Dados

Esta seção apresenta os elementos referentes à programação interna ao Banco de Dados. Obrigatoriamente, é necessário a construção de:

1. Duas *stored procedures*;
2. Uma *trigger*;
3. Uma visão.

3.1 *Stored Procedures*

Foram criadas três *stored procedures*, duas para consultas simples e uma para a consulta avançada.

As *stored procedures* para as consultas básicas retornam uma tabela com os campos: **código da natureza**, **descrição da natureza** e o **total gasto por natureza**, se a consulta for consulta simples por natureza (Listagem 5) ou código do programa, descrição do programa e o total gasto por programa, se a consulta simples for por programa (Listagem 6).

Os parâmetros das consultas simples foram alteradas das consultas originais, passando a ter além de município, o nome de uma natureza de despesa ou o nome de um programa, dependendo da consulta realizada.

As *stored procedures* para consultas simples tratam o caso de o usuário não digitar um município, assim insere um valor *default* para cidade, sendo esta Campinas.

Caso o usuário não insira um valor para natureza de despesa ou programa, a consulta retornará o total gasto para todas as naturezas de despesa ou programas. Caso insira, retornará o total gasto com uma natureza específica ou programa específico.

```
1  CREATE OR REPLACE FUNCTION CONSULTA_SIMPLES_NATUREZA(VARCHAR(100),
       VARCHAR (100), VARCHAR(100), VARCHAR (100))
2  RETURNS TABLE(cod INTEGER, descricao VARCHAR(100), gasto NUMERIC)
       AS $$
3  DECLARE
4    natDesc ALIAS FOR $1;
5    cidade ALIAS FOR $2;
6    tipo_consulta ALIAS FOR $3;
7    texto_consulta ALIAS FOR $4;
8  BEGIN
9    IF cidade IS NULL THEN
10     cidade := 'Campinas';
11   END IF;
```

```
12
13   IF natDesc IS NULL THEN --RETORNA A AGREGACAO PARA TODAS AS
       NATUREZAS
14     RETURN QUERY
15     SELECT N.codigo, N.descricao, SUM(Desp.valor) AS gasto FROM
         despesa Desp, natureza N, (
16       SELECT M.codigo FROM CodMunicipio M WHERE M.descricao = cidade
           )Mun --usando a view
17     WHERE Mun.codigo = Desp.codigomunicipio AND N.codigo = Desp.
         codigonatureza
18     GROUP BY N.codigo, N.descricao ORDER BY gasto DESC;
19   ELSE -- RETORNA A AGREGACAO PARA UMA NATUREZA ESPECIFICA
20     RETURN QUERY
21     SELECT N.codigo, N.descricao, SUM(Desp.valor) AS gasto FROM
         despesa Desp,
22       (SELECT * FROM naturezaView Nat WHERE Nat.descricao ILIKE
           natDesc )N, --usando view
23       (SELECT M.codigo FROM CodMunicipio M WHERE M.descricao =
           cidade)Mun --usando a view
24     WHERE Mun.codigo = Desp.codigomunicipio AND N.codigo = Desp.
         codigonatureza
25     GROUP BY N.codigo, N.descricao ORDER BY gasto DESC;
26   END IF;
27
28   INSERT INTO HISTORICO VALUES(1, tipo_consulta, texto_consulta);
29 END;
30 $$ LANGUAGE plpgsql;
```

Listing 5: *Stored Procedure* de Consulta Simples por Natureza

```
1 CREATE OR REPLACE FUNCTION CONSULTA_SIMPLES_PROGRAMA(VARCHAR(100),
    VARCHAR (100), VARCHAR(100), VARCHAR(100))
2 RETURNS TABLE(cod INTEGER, descricao VARCHAR(100), gasto NUMERIC)
    AS $$
3 DECLARE
4   progDesc ALIAS FOR $1;
5   cidade ALIAS FOR $2;
6   tipo_consulta ALIAS FOR $3;
7   texto_consulta ALIAS FOR $4;
8
9 BEGIN
```

```
10   IF cidade IS NULL THEN
11     cidade := 'Campinas';
12   END IF;
13
14   IF progDesc IS NULL THEN --RETORNA A AGREGACAO PARA TODAS OS
       PROGRAMAS
15     RETURN QUERY
16     SELECT P.codigo, P.descricaointernamunicipio, SUM(Desp.valor) AS
         gasto FROM despesa Desp, programa P,
17       (SELECT M.codigo FROM CodMunicipio M WHERE M.descricao =
           cidade)Mun --usando view
18     WHERE Mun.codigo = Desp.codigomunicipio AND P.codigo = Desp.
         codigoprograma
19     GROUP BY P.codigo, P.descricaointernamunicipio ORDER BY gasto
         DESC;
20   ELSE -- RETORNA A AGREGACAO PARA UM PROGRAMA ESPECIFICA
21     RETURN QUERY
22     SELECT P.codigo, P.descricao, SUM(Desp.valor) AS gasto FROM
         despesa Desp,
23       (SELECT * FROM ProgramaView Prog WHERE Prog.descricao ILIKE
           progDesc)P, --usando view
24       (SELECT M.codigo FROM CodMunicipio M WHERE M.descricao =
           cidade)Mun --usando view
25     WHERE Mun.codigo = Desp.codigomunicipio AND P.codigo = Desp.
         codigoprograma
26     GROUP BY P.codigo, P.descricao ORDER BY gasto DESC;
27   END IF;
28
29   INSERT INTO HISTORICO VALUES(1, tipo_consulta, texto_consulta);
30 END;
31 $$ LANGUAGE plpgsql;
```

Listing 6: *Stored Procedure* de Consulta Simples por Programa

Já a *stored procedure* para a consulta avançada (listagem 7) retorna uma tabela com os seguintes campos: **nome do programa**, **nome da natureza** e o **total gasto**. Os parâmetros da consulta são natureza1, natureza2 (o usuário tem a opção de inserir uma natureza de despesa ou duas), o município, intervalo de tempo com ano inicial e ano final e um intervalo de valores com valor inicial e valor final.

Caso o usuário não insira algum dado de intervalo de tempo (ano inicial ou ano final), é inserido um valor *default* com o ano mais antigo que está presente no banco ou

ano mais recente.

Caso o limite inferior de valor não seja designado, a *stored procedure* insere o valor *default* 0 (zero). Além disso, trata cada uma das seguintes situações:

1. Natureza2 não é informada e limite superior não é informado;
2. Natureza2 não é informada e limite superior não é informado;
3. Natureza2 é informada e limite superior é informado;
4. Natureza2 é informada e limite superior é informado.

Para cada uma dessas situações é retornada uma consulta diferente, específica para a situação.

```
1  CREATE OR REPLACE FUNCTION CONSULTA_AVANCADA(VARCHAR(100), VARCHAR
       (100), VARCHAR(100), INTEGER, INTEGER, REAL, REAL, VARCHAR(100),
        VARCHAR(100))
2  RETURNS TABLE(descricaoPrograma VARCHAR(100), descricaoNatureza
       VARCHAR(100), gasto NUMERIC) AS $$
3  DECLARE
4    natureza1 ALIAS FOR $1;
5    natureza2 ALIAS FOR $2;
6    cidade ALIAS FOR $3;
7    anoInic ALIAS FOR $4;
8    anoFinal ALIAS FOR $5;
9    limiteInferior ALIAS FOR $6;
10   limiteSuperior ALIAS FOR $7;
11   tipo_consulta ALIAS FOR $8;
12   texto_consulta ALIAS FOR $9;
13 BEGIN
14   IF anoInic IS NULL THEN
15     SELECT dataano INTO anoInic FROM despesa ORDER BY dataano ASC
           LIMIT 1;
16   END IF;
17
18   IF anoFinal IS NULL THEN
19     SELECT dataano INTO anoFinal FROM despesa ORDER BY dataano DESC
           LIMIT 1;
20   END IF;
21
22   IF limiteInferior IS NULL THEN
```

```
23    limiteInferior := 0;
24  END IF;
25
26  IF cidade IS NULL THEN
27    cidade := 'Campinas';
28  END IF;
29
30  IF natureza2 IS NULL AND limiteSuperior IS NULL THEN
31    RETURN QUERY
32    SELECT P.descricaointernamunicipio , N.descricao , SUM(D.valor)
          AS gasto FROM despesa D, programa P,
33      (SELECT * FROM naturezaView Nat WHERE Nat.descricao ILIKE
            natureza1) N, --usando view
34      (SELECT M.codigo FROM CodMunicipio M WHERE M.descricao =
            cidade)Mun --usando view
35    WHERE Mun.codigo = D.codigomunicipio AND D.codigoprograma = P.
          codigo AND D.codigonatureza = N.codigo AND
36      (dataano >= anoInic and dataano <= anoFinal)
37    GROUP BY P.descricaointernamunicipio , N.descricao
38    HAVING SUM(D.valor) > limiteInferior
39    ORDER BY P.descricaointernamunicipio ASC;
40
41  ELSIF natureza2 IS NULL AND limiteSuperior IS NOT NULL THEN
42    RETURN QUERY
43    SELECT P.descricaointernamunicipio , N.descricao , SUM(D.valor)
          AS gasto FROM despesa D, programa P,
44      (SELECT * FROM naturezaView Nat WHERE Nat.descricao ILIKE
            natureza1) N, --usando view
45      (SELECT M.codigo FROM CodMunicipio M WHERE M.descricao =
            cidade)Mun --usando view
46    WHERE Mun.codigo = D.codigomunicipio AND D.codigoprograma = P.
          codigo AND D.codigonatureza = N.codigo AND
47      (dataano >= anoInic and dataano <= anoFinal)
48    GROUP BY P.descricaointernamunicipio , N.descricao
49    HAVING SUM(D.valor) > limiteInferior AND SUM(D.valor) <
          limiteSuperior
50    ORDER BY P.descricaointernamunicipio ASC;
51
52  ELSIF natureza2 IS NOT NULL AND limiteSuperior IS NULL THEN
53    RETURN QUERY
```

```
54    SELECT P.descricaointernamunicipio , N.descricao , SUM(D.valor)
          AS gasto FROM despesa D, programa P,
55      (SELECT * FROM naturezaView Nat WHERE Nat.descricao ILIKE
            natureza1 UNION --view
56      SELECT * FROM naturezaView Nat WHERE Nat.descricao ILIKE
            natureza2) N, --view
57      (SELECT M.codigo FROM CodMunicipio M WHERE M.descricao =
            cidade)Mun --view
58    WHERE Mun.codigo = D.codigomunicipio AND D.codigoprograma = P.
          codigo AND D.codigonatureza = N.codigo AND
59      (dataano >= anoInic and dataano <= anoFinal)
60    GROUP BY P.descricaointernamunicipio , N.descricao
61    HAVING SUM(D.valor) > limiteInferior
62    ORDER BY P.descricaointernamunicipio ASC;
63
64  ELSE
65    RETURN QUERY
66    SELECT P.descricaointernamunicipio , N.descricao , SUM(D.valor)
          AS gasto FROM despesa D, programa P,
67      (SELECT * FROM naturezaView Nat WHERE Nat.descricao ILIKE
            natureza1 UNION --view
68      SELECT * FROM naturezaView Nat WHERE Nat.descricao ILIKE
            natureza2) N, --view
69      (SELECT M.codigo FROM CodMunicipio M WHERE M.descricao =
            cidade)Mun --view
70    WHERE Mun.codigo = D.codigomunicipio AND D.codigoprograma = P.
          codigo AND D.codigonatureza = N.codigo AND
71      (dataano >= anoInic and dataano <= anoFinal)
72    GROUP BY P.descricaointernamunicipio , N.descricao
73    HAVING SUM(D.valor) > limiteInferior AND SUM(D.valor) <
          limiteSuperior
74    ORDER BY P.descricaointernamunicipio ASC;
75  END IF;
76
77  INSERT INTO HISTORICO VALUES(1, tipo_consulta, texto_consulta);
78 END;
79 $$ LANGUAGE plpgsql;
```

Listing 7: *Stored Procedure* de Consulta Avançada

3.2 *Trigger*

Para a construção da *trigger* foi necessária a criação de uma tabela **HISTORICO** como mostra a Listagem 8, onde é inserido o tipo de consulta realizada e a data e horário da consulta. O gatilho é a inserção do valor de tipo de consulta na tabela.

```
1 CREATE TABLE HISTORICO (
2   COD SERIAL NOT NULL PRIMARY KEY,
3   TIPO_CONSULTA VARCHAR(100),
4   TEXTO_CONSULTA VARCHAR(100),
5   DATA DATE,
6   HORARIO TIME
7 );
```

Listing 8: Tabela Criada para a Utilização da Trigger

Na Listagem 9 apresentamos o código da *stored procedure* associada à *trigger*, cuja responsabilidade é realizar a inserção da data e horário em que a consulta foi realizada, assim como tratar a inserção sequencial do `codigo` (chave primária) da tupla.

```
1  CREATE OR REPLACE FUNCTION HISTORICO_F() RETURNS trigger AS
       $HISTORICO_F$
2  DECLARE valAntigo INTEGER;
3  BEGIN
4    SELECT COUNT(COD) INTO valAntigo FROM HISTORICO;
5    IF (valAntigo != 0) THEN
6      SELECT MAX(COD) INTO valAntigo FROM HISTORICO;
7      NEW.COD := valAntigo+1;
8    END IF;
9
10   NEW.DATA := current_date;
11   NEW.HORARIO := current_time;
12   RETURN NEW;
13 END;
14 $HISTORICO_F$ LANGUAGE plpgsql;
15
16 CREATE TRIGGER HISTORICO_F BEFORE INSERT ON HISTORICO
17 FOR EACH ROW EXECUTE PROCEDURE HISTORICO_F();
```

Listing 9: *Trigger* e *Stored Procedure* Associada

3.3 Visão

Foram criadas quatro visões, uma para selecionar os códigos e nomes dos municípios, assim como a seleção de códigos e nomes de programas, outra para selecionar códigos e nomes de naturezas de despesa,e por último, foi acrescentada uma visão para a consulta avançada que seleciona o código do município, o código do programa, código da natureza, dataano e o valor total gasto, pois estas são subconsultas recorrentes em todas as consultas.

```
1 CREATE VIEW CodMunicipio(codigo) AS
2 SELECT codigo, descricao
3 FROM municipio;
```

Listing 10: Visão para Município

```
1 CREATE VIEW NaturezaView AS
2 SELECT codigo, descricao
3 FROM natureza;
```

Listing 11: Visão para Natureza de Despesa

```
1 CREATE VIEW ProgramaView(codigo, descricao) AS
2 SELECT codigo, descricaointernamunicipio
3 FROM programa;
```

Listing 12: Visão para Programa

A visão apresentada na Listagem 13 não foi utilizada no projeto, pois com ela, a consulta levou cerca de 17 segundos para retornar resultados, enquanto anteriormente, sem a visão, levava 194 milissegundos.

```
1 CREATE VIEW ConsultaAvancada(munic, natu, progr, data, gasto) AS
2   SELECT d.codigomunicipio, d.codigoprograma, d.codigonatureza, d.
      dataano, SUM(d.valor)
3   FROM despesa d, programa p, natureza n, municipio m WHERE m.codigo
       = d.codigomunicipio
4   AND d.codigoprograma = p.codigo AND d.codigonatureza = n.codigo
5   GROUP BY d.codigomunicipio, d.codigoprograma, d.codigonatureza, d.
      dataano;
```

Listing 13: Visão para Consulta Avançada

4 Outras Informações sobre o Projeto

No início do desenvolvimento utilizamos técnicas aprendidas em ISI, ES1 e ES2 para elicitação de requisitos para o projeto, utilização do cronograma com os requisistos e o controle semanal com o *Status Report*, atualizando o que foi feito e o que faltava para que o sistema ficasse completo.

Além disso, nos aspectos técnicos, utilizamos o Netbeans como ambiente de desenvolvimento, a linguagem de programação JAVA, o PostgreSQL como SGDB e a ferramenta GIT em conjunto com o Github para realizar o controle de versões e também, paralelizar o desenvolvimento entre os membros.

Para auxiliar o desenvolvimento do projeto, utilizamos a arquitetura MVC, em que consiste dividir o sistema em: *Model*, *View* e *Control*. Sendo o *Model* responsável pelas características de cada consulta, o *View* pela representação dos dados na tela e o *Control* sendo a interface do sistema com o SGDB (que executaria as consultas).

Foi utilizado também o *driver* JDBC para realizar a conexão com o SGDB.

A tela principal do programa consiste em:

1. Um banner para o logo da empresa;
2. Um menu à esquerda com as opções de consulta;
3. No meio o formulário para realizar a consulta, ou então, para exibição de resultados das consultas;
4. No rodapé o ano em que foi desenvolvido;

Na consulta normal o usuário deverá escolher se a consulta é por Natureza ou Programa, logo após, deverá escolher o Município em que a consulta englobará, e por último no campo de texto o título de algum Programa ou Natureza. Quando campo de texto está vazio, o sistema irá realizar a consulta retornando todos os programas/naturezas. Quando não há resultados, é exibido a mensagem como na Figura 8.

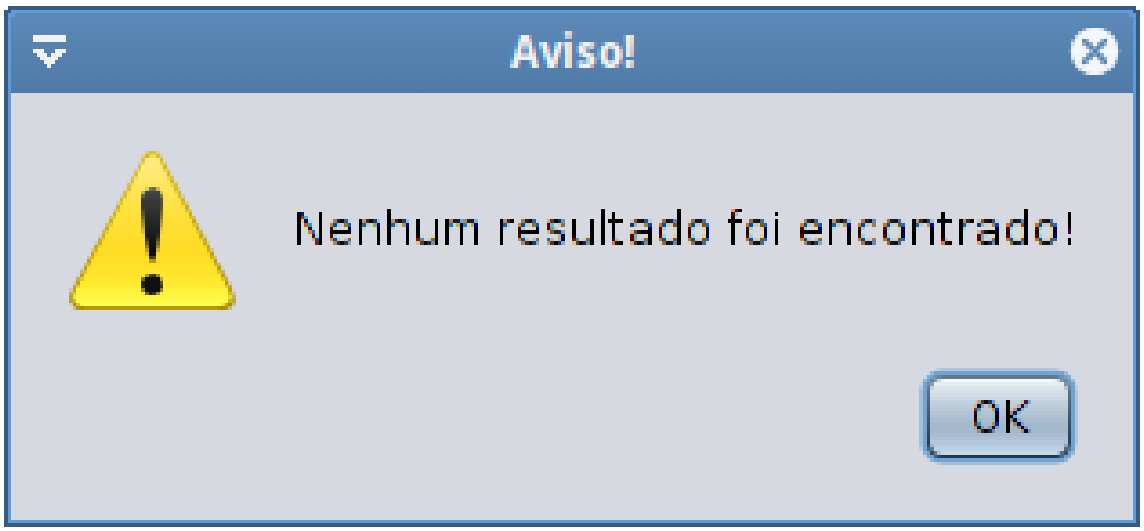

Figura 8: Mensagem de aviso

Para realizar a consulta avançada, o usuário deverá escolher o Município em que a pesquisa englobará, e pelo menos uma Natureza (caso não entre com nenhuma Natureza, é destacado que é necessário preencher). Com os campos preenchidos, a consulta poderá retornar os resultados exibindo-o em uma tabela com os campos de Município, Descrição da natureza e o total de Gasto feitos; caso não exista nenhum resultado, é exibido a mesma mensagem da consulta normal (Figura 8).

Para exibir o histórico de consultas o usuário deve apenas ativar a função, não tendo mais nenhum outro trabalho posterior. Caso a pesquise retorne resultados, é exibido uma tabela com o tipo de consulta, os campos de pesquisa, a data e o horário que foi realizado a consulta; caso contrário, é exibido uma tabela vazia.

5 Considerações Finais

Como explicado anteriormente, encontramos dificuldades com a criação da visão sobre despesa na consulta avançada, pois com ela, a consulta levava muito mais tempo para retornar resultados do que levava anteriormente sem ela.

Podemos notar isso com os planos de consulta da mesma consulta avançada, uma utilizando a visão, Figura 9, e a outra não, Figura 10:

```
Sort  (cost=468849.71..468851.42 rows=684 width=468) (actual time=19692.793..19692.813 rows=600 loops=1)
  Sort Key: (sum(d.valor))
  Sort Method:  quicksort  Memory: 160kB
  ->  HashAggregate  (cost=468810.66..468817.50 rows=684 width=468) (actual time=19692.114..19692.201 rows=600 loops=1)
        ->  Nested Loop  (cost=4412.48..295529.52 rows=23104152 width=468) (actual time=69.861..4455.023 rows=20672136 loops=1)
              ->  Seq Scan on despesa d  (cost=0.00..2297.78 rows=33778 width=0) (actual time=0.052..39.072 rows=33778 loops=1)
              ->  Materialize  (cost=4412.48..4431.55 rows=684 width=468) (actual time=0.002..0.042 rows=612 loops=33778)
                    ->  Nested Loop  (cost=4412.48..4428.13 rows=684 width=468) (actual time=69.802..70.027 rows=612 loops=1)
                          ->  HashAggregate  (cost=4412.48..4415.04 rows=114 width=24) (actual time=69.747..69.772 rows=51 loops=1)
                                Filter: ((sum(d.valor) > 0::numeric) AND (sum(d.valor) < 500000000000000::numeric))
                                ->  Nested Loop  (cost=2.56..3821.36 rows=33778 width=24) (actual time=0.050..48.578 rows=33778 loops=1)
                                      Join Filter: (d.codigomunicipio = m.codigo)
                                      ->  Seq Scan on municipio m  (cost=0.00..1.01 rows=1 width=4) (actual time=0.003..0.003 rows=1 loops=1)
                                      ->  Hash Join  (cost=2.56..3398.13 rows=33778 width=24) (actual time=0.042..43.288 rows=33778 loops=1)
                                            Hash Cond: (d.codigonatureza = n.codigo)
                                            ->  Hash Join  (cost=1.14..2932.25 rows=33778 width=24) (actual time=0.027..34.055 rows=33778 loops=1)
                                                  Hash Cond: (d.codigoprograma = p.codigo)
                                                  ->  Seq Scan on despesa d  (cost=0.00..2466.67 rows=33778 width=24) (actual time=0.007..23.717 rows=33778 loops=1)
                                                        Filter: ((dataano >= 2000) AND (dataano <= 2015))
                                                  ->  Hash  (cost=1.06..1.06 rows=6 width=4) (actual time=0.009..0.009 rows=6 loops=1)
                                                        Buckets: 1024  Batches: 1  Memory Usage: 1kB
                                                        ->  Seq Scan on programa p  (cost=0.00..1.06 rows=6 width=4) (actual time=0.003..0.005 rows=6 loops=1)
                                            ->  Hash  (cost=1.19..1.19 rows=19 width=4) (actual time=0.009..0.009 rows=19 loops=1)
                                                  Buckets: 1024  Batches: 1  Memory Usage: 1kB
                                                  ->  Seq Scan on natureza n  (cost=0.00..1.19 rows=19 width=4) (actual time=0.003..0.006 rows=19 loops=1)
                          ->  Materialize  (cost=0.00..3.41 rows=6 width=436) (actual time=0.001..0.003 rows=12 loops=51)
                                ->  Nested Loop  (cost=0.00..3.38 rows=6 width=436) (actual time=0.048..0.112 rows=12 loops=1)
                                      ->  Nested Loop  (cost=0.00..2.26 rows=1 width=218) (actual time=0.042..0.101 rows=2 loops=1)
                                            ->  Seq Scan on natureza  (cost=0.00..1.24 rows=1 width=218) (actual time=0.021..0.074 rows=2 loops=1)
                                                  Filter: ((descricao)::text ~~* '%outros%'::text)
                                            ->  Seq Scan on municipio m  (cost=0.00..1.01 rows=1 width=0) (actual time=0.003..0.003 rows=1 loops=2)
                                            ->  Seq Scan on municipio m  (cost=0.00..1.01 rows=1 width=0) (actual time=0.003..0.003 rows=1 loops=2)
                                                  Filter: ((m.descricao)::text = 'Campinas'::text)
                                      ->  Seq Scan on programa p  (cost=0.00..1.06 rows=6 width=218) (actual time=0.001..0.001 rows=6 loops=2)
Total runtime: 19693.197 ms
```

Figura 9: Consulta avançada com visão sobre despesa

```
Sort  (cost=2173.64..2173.66 rows=6 width=444) (actual time=166.868..166.868 rows=11 loops=1)
  Sort Key: (sum(d.valor))
  Sort Method:  quicksort  Memory: 27kB
  ->  HashAggregate  (cost=2173.43..2173.56 rows=6 width=444) (actual time=166.842..166.847 rows=11 loops=1)
        Filter: ((sum(d.valor) > 0::numeric) AND (sum(d.valor) < 500000000000000::numeric))
        ->  Nested Loop  (cost=59.17..2151.20 rows=1778 width=444) (actual time=80.923..160.846 rows=6618 loops=1)
              Join Filter: (d.codigomunicipio = municipio.codigo)
              ->  Seq Scan on municipio  (cost=0.00..1.01 rows=1 width=4) (actual time=0.012..0.013 rows=1 loops=1)
                    Filter: ((descricao)::text = 'Campinas'::text)
              ->  Hash Join  (cost=59.17..2127.97 rows=1778 width=448) (actual time=80.901..159.706 rows=6618 loops=1)
                    Hash Cond: (d.codigoprograma = p.codigo)
                    ->  Nested Loop  (cost=58.04..2102.38 rows=1778 width=234) (actual time=80.880..157.269 rows=6618 loops=1)
                          ->  Seq Scan on natureza  (cost=0.00..1.24 rows=1 width=222) (actual time=0.021..0.095 rows=2 loops=1)
                                Filter: ((descricao)::text ~~* '%outros%'::text)
                          ->  Bitmap Heap Scan on despesa d  (cost=58.04..2078.92 rows=1778 width=20) (actual time=67.657..77.983 rows=3309 loops=2
                                Recheck Cond: (d.codigonatureza = natureza.codigo)
                                Filter: ((d.dataano >= 0) AND (d.dataano <= 2015))
                                ->  Bitmap Index Scan on indice_codnatureza  (cost=0.00..57.59 rows=1778 width=0) (actual time=67.470..67.470 rows=
                                      Index Cond: (d.codigonatureza = natureza.codigo)
                    ->  Hash  (cost=1.06..1.06 rows=6 width=222) (actual time=0.009..0.009 rows=6 loops=1)
                          Buckets: 1024  Batches: 1  Memory Usage: 1kB
                          ->  Seq Scan on programa p  (cost=0.00..1.06 rows=6 width=222) (actual time=0.005..0.006 rows=6 loops=1)
Total runtime: 183.738 ms
```

Figura 10: Consulta avançada sem visão sobre despesa

A consulta com a visão sobre a despesa agrega todos os valores e depois filtra, já a consulta sem a visão filtra o máximo para diminuir o número de tuplas e depois agrega, deixando assim a consulta otimizada.

A

Planos de Consulta para Calcular Tabela 1

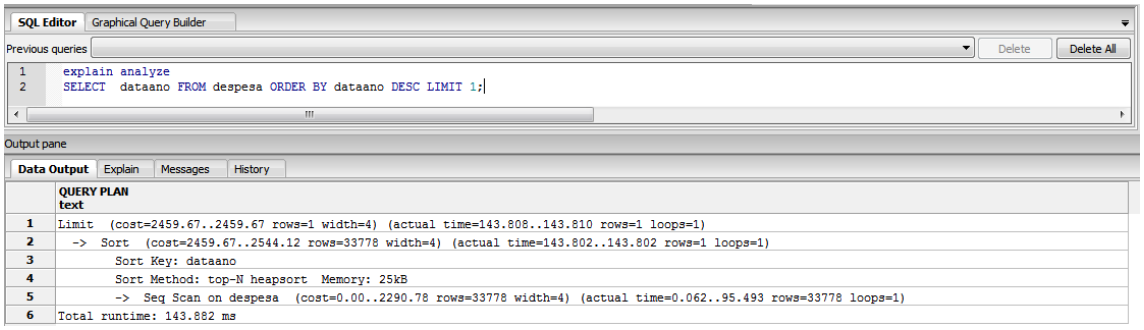

Figura 11: Consulta auxiliar em dataano

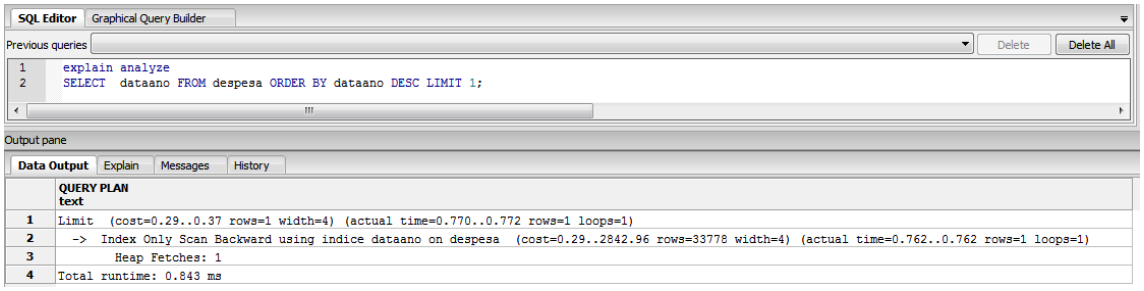

Figura 12: Consulta auxiliar em dataano com Índice

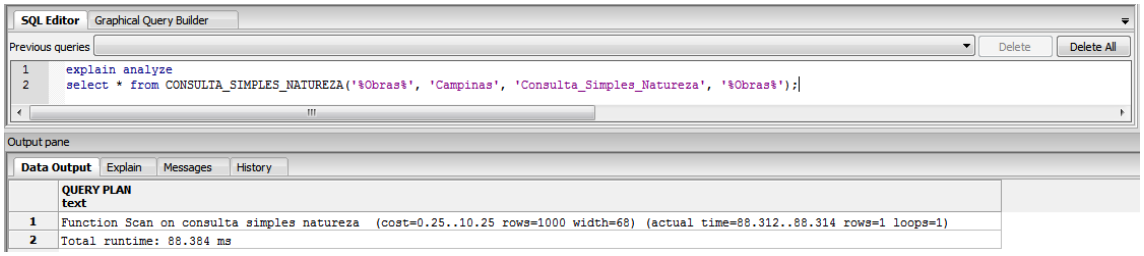

Figura 13: Medição Consulta 1: Informando natureza específica

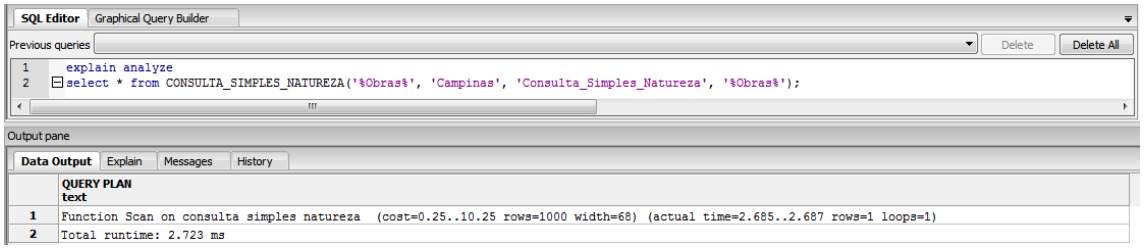

Figura 14: Medição Consulta 1: Informando natureza específica com Índice

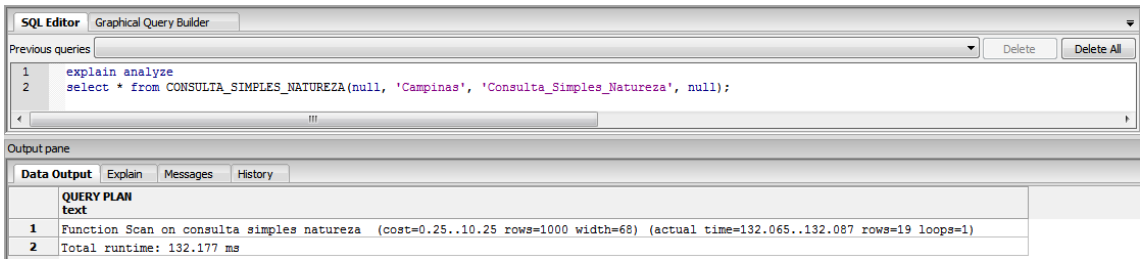

Figura 15: Medição Consulta 1: Agregando todas as naturezas

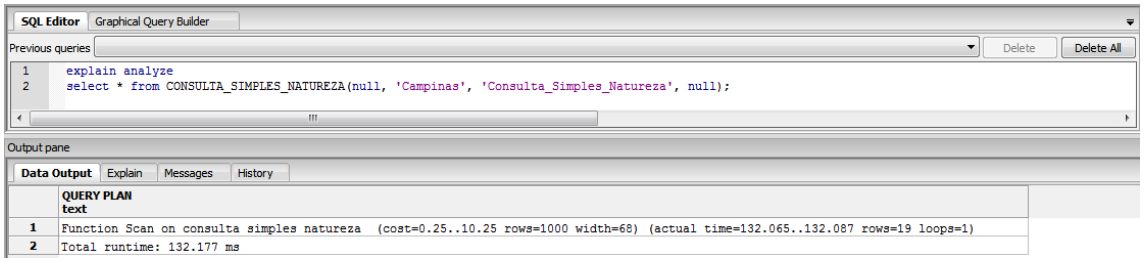

Figura 16: Medição Consulta 1: Agregando todas as naturezas, com Índice

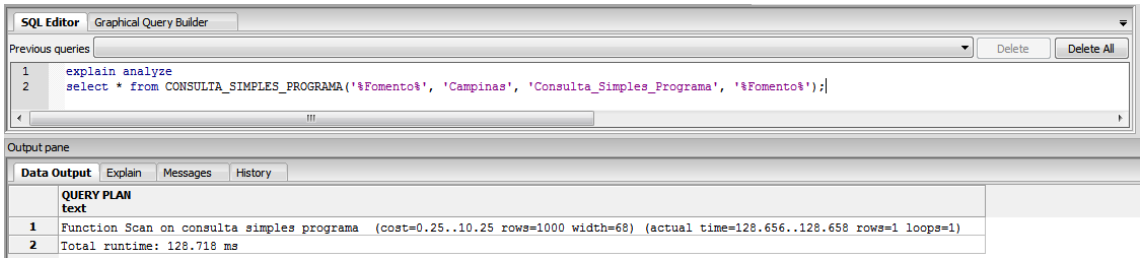

Figura 17: Medição Consulta 2: Informando programa específico

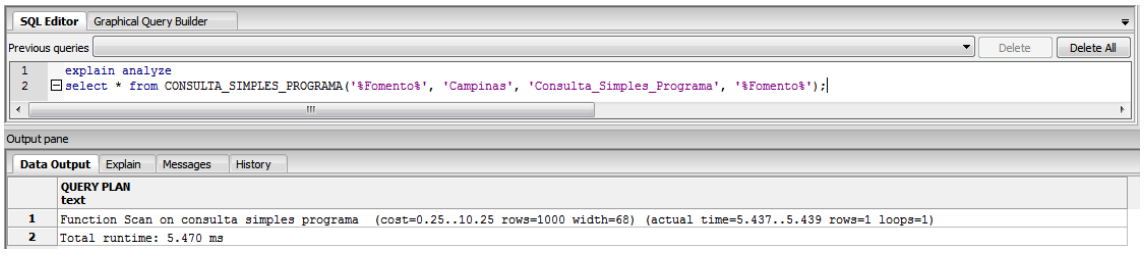

Figura 18: Medição Consulta 2: Informando programa específico com Índice

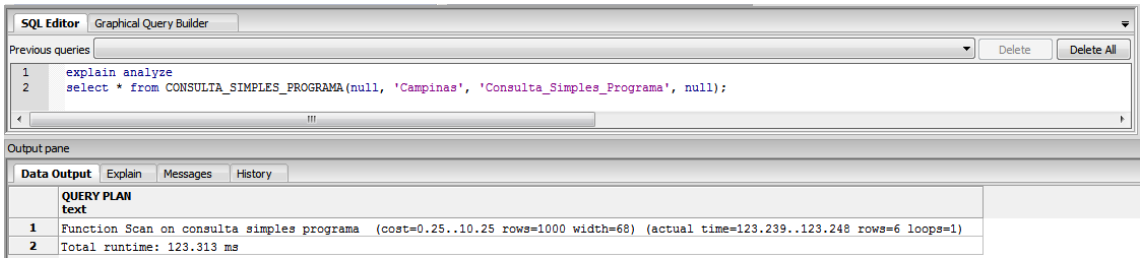

Figura 19: Medição Consulta 2: Agregando todos os programas

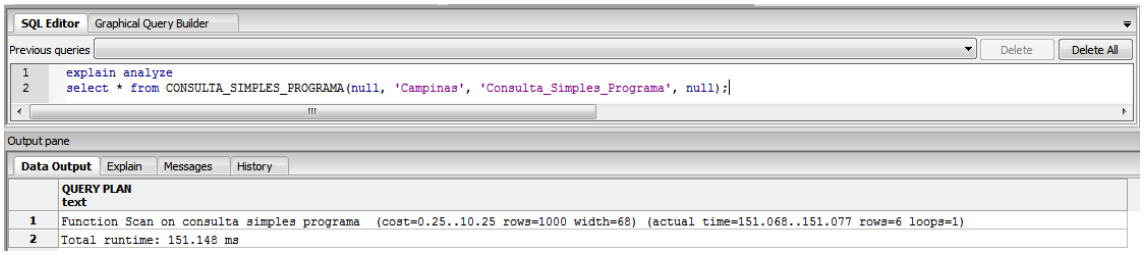

Figura 20: Medição Consulta 2: Agregando todos os programas, com Índice