



UNIVERSIDADE DE SÃO PAULO
CAMPUS DE SÃO CARLOS
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO (ICMC)

Trabalho Prático T6

Eduardo Costa Miranda Azevedo - 12677151

Gustavo de Oliveira Martins - 12625531

Ivan Barbosa Pinheiro - 9050552

Michelle Schmitt Gmurczyk - 9424315

São Carlos

2023



UNIVERSIDADE DE SÃO PAULO

CAMPUS DE SÃO CARLOS

INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO (ICMC)

Trabalho Prático T6

Relatório do projeto **Trabalho Prático T5 – server side PIPgSQL** dado como exigência para obtenção de menção na disciplina de **Laboratório Bases de Dados (SCC0541)**, sob orientação do William Carlos Giovanetti Gomes e do Prof. Dr. Caetano Traina Júnior.

Sumário

1 - Introdução	5
2 - Objetivos	5
3 - Respostas das questões	6
3.1 - Exercício 1	6
3.1.1 - Trigger TR_Airports e função VerificaAeroporto() criadas:	6
3.1.2 - Tentando inserir em uma “Cidade Inexistente”	7
3.1.3 - Inserindo um aeroporto em São Carlos para testar a atualização:	7
3.1.4 - “Transferindo” o aeroporto para “Nárnia”:	7
3.2 - Exercício 2 (a b c)	8
Criação da função, gatilho e ponderações iniciais	9
3.2.a - Inserção	10
3.2.a.1) Trigger e função criadas:	10
Código de validação	10
3.2.a.2) Testes:	10
Passo 1:	10
Passo 2:	11
Passo 3:	11
3.2.b - Deleção	11
3.2.b.1) Trigger e função criadas:	11
Código de validação	11
3.2.b.2) Testes:	12
Passo 1:	12
Passo 2:	12
Passo 3:	13
3.2.b - Alteração	13
3.2.b.1) Trigger e função criadas:	13
Código de validação	13
3.2.b.2) Testes:	14
Passo 1:	14
Passo 2:	14
Passo 3:	15
3.2 - Exercício 2 (d)	17
d.1) Trigger e função criadas:	17
d.2) Tentando inserir um resultado com StatusId negativo:	17

1 - Introdução

Este trabalho visa nos familiarizar com os conceitos de triggers atrelados à funções no postgresql.

2 - Objetivos

Desenvolver nossas habilidades técnicas e interpretativas para resolver problemas de banco de dados com as características exigidas pelos enunciados, isto é, tarefas que precisam de gatilhos específicos ao incluir, alterar ou excluir dados de tabelas.

3 - Respostas das questões

3.1 - Exercício 1

Exercício 1) Na tabela **AIRPORTS**, há vários Aeroportos cujo atributo **City** apresenta um nome que não aparece na tabela **GEOCITIES15K** no atributo **name**. A equipe responsável pelo projeto decidiu que, em relação aos dados já persistidos, nada irá ser feito. No entanto, para novos aeroportos inseridos na base, será obrigatório vinculá-los a uma cidade. Sua equipe foi designada para desenvolver uma *trigger* que avalia se o atributo **City** de um aeroporto sendo inserido (ou modificado) na tabela **AIRPORTS** não encontra correspondência com o atributo **Name** da tabela **GEOCITIES15K**. Quando isso acontecer, a operação não deve ser concluída e uma exceção deve ser lançada.

- (a) Nome da função: **VerificaAeroporto**
- (b) Nome do *trigger*: **TR_Airports**
- (c) Mensagem da exceção: **'Cidade não encontrada! Operação cancelada.'**

Faça ao menos um teste para inserção e um para atualização, e mostre-os no relatório junto do resultado.

3.1.1 - Trigger TR_Airports e função VerificaAeroporto() criadas:

```
● ----- Exercício 1
CREATE OR REPLACE FUNCTION VerificaAeroporto()
  RETURNS TRIGGER AS
  $$
  BEGIN
    IF NEW.city IS NOT NULL AND
      NOT EXISTS (SELECT 1 FROM GEOCITIES15K gc WHERE gc.name = NEW.city)
      THEN RAISE EXCEPTION 'Cidade não encontrada! Operação cancelada.';
    RETURN NULL;
    END IF;
    RETURN NEW;
  END;
  $$ LANGUAGE plpgsql;

● CREATE TRIGGER TR_Airports
  BEFORE INSERT OR UPDATE ON AIRPORTS
  FOR EACH ROW
  EXECUTE FUNCTION VerificaAeroporto();
```

3.1.2 - Tentando inserir em uma “Cidade Inexistente”

```
●INSERT INTO Airports (Ident, Type, Name, LatDeg, LongDeg, ElevFt, Continent, ISOCountry, ISORegion, City, Scheduled_service, GPSCode, IATACode, LocalCode, HomeLink, WikipediaLink, Keywords)
VALUES ('00024A', 'heliport', 'Total Rf Heliport', 40.07080078125, -74.93360137939453, 11, 'NA', 'US', 'US-PA',
      'CIDADE INEXISTENTE',
      'no', '00A', '00A', '', '', '', '');
```

Estadísticas 1 X

Erro SQL [P0001]: ERROR: Cidade não encontrada! Operação cancelada.
Onde: PL/pgSQL function verificaaeroporto() line 5 at RAISE

Detalhes >>

```
INSERT INTO Airports (Ident, Type, Name, LatDeg, LongDeg, ElevFt, Continent, ISOCountry, ISORegion, City, Scheduled_service, GPSCode, IATACode, LocalCode, HomeLink, WikipediaLink, Keywords)
VALUES ('00024A', 'heliport', 'Total Rf Heliport', 40.07080078125, -74.93360137939453, 11, 'NA', 'US', 'US-PA',
      'CIDADE INEXISTENTE',
      'no', '00A', '00A', '', '', '', '');
```

3.1.3 - Inserindo um aeroporto em São Carlos para testar a atualização:

```
●INSERT INTO Airports (Ident, Type, Name, LatDeg, LongDeg, ElevFt, Continent, ISOCountry, ISORegion, City, Scheduled_service, GPSCode, IATACode, LocalCode, HomeLink, WikipediaLink, Keywords)
VALUES ('00025A', 'heliport', 'Total Rf Heliport', 40.07080078125, -74.93360137939453, 11, 'NA', 'US', 'US-PA',
      'São Carlos',
      'no', '00A', '00A', '', '', '', '');
```

Estadísticas 1 X

Name	Value
Updated Rows	1
Query	INSERT INTO Airports (Ident, Type, Name, LatDeg, LongDeg, ElevFt, Continent, ISOCountry, ISORegion, City, Scheduled_service, GPSCode, IATACode, LocalCode, HomeLink, WikipediaLink, Keywords) VALUES ('00025A', 'heliport', 'Total Rf Heliport', 40.07080078125, -74.93360137939453, 11, 'NA', 'US', 'US-PA', 'São Carlos', 'no', '00A', '00A', '', '', '', '')
Start time	Mon May 29 21:15:36 BRT 2023
Finish time	Mon May 29 21:15:36 BRT 2023

3.1.4 - “Transferindo” o aeroporto para “Nárnia”:

```
SELECT * FROM airports a
```

```
●UPDATE Airports
  SET City = 'Nárnia'
 WHERE Ident = '00025A';
```

Estadísticas 1 X

Erro SQL [P0001]: ERROR: Cidade não encontrada! Operação cancelada.
Onde: PL/pgSQL function verificaaeroporto() line 5 at RAISE

Posição do erro:

3.2 - Exercício 2 (a b c)

Exercício 2) 2. Execute o SEGUINTE *script* e faça o que se pede.

```
CREATE TABLE Results_Status (  
    StatusID INTEGER PRIMARY KEY,  
    Contagem INTEGER,  
    FOREIGN KEY (StatusID) REFERENCES Status(StatusID)  
);  
  
INSERT INTO Results_Status  
    SELECT S.StatusId , COUNT (*)  
    FROM Status S JOIN Results R ON R.StatusID = S.StatusID  
    GROUP BY S.StatusId , S.Status;
```

RESPONDA: O que esse *script* faz?

O script acima cria uma tabela chamada Results_Status, com os atributos StatusID e Contagem. Sobre esses atributos, podemos destrinchar que: StatusID representa a chave primária simples dessa tabela, e tem “INTEGER” como seu tipo de dado. Além disso, o atributo StatusID tem como referência o atributo StatusID da tabela já criada “STATUS”, o que indica que a chave primária da tabela results_status é também uma chave estrangeira; Contagem também tem “INTEGER” como tipo de dado e é um atributo simples, não único e poderá ter valor nulo.

Depois da criação, temos um trecho em que ocorre uma inserção na tabela results_status. Essa inserção, por sua vez, nos indica que(por conhecer a estrutura das tabelas STATUS e RESULTS) ela está trazendo o id de cada status no atributo “StatusID” e quantas vezes aquele tipo de status aparece na tabela resultados, no atributo “Contagem”. Essa inserção torna mais simples trazer métricas relacionadas a quantas vezes cada tipo de Status aconteceu em todos os resultados mapeados na tabela Results.

Depois, crie uma única *trigger* para as questões (a), (b) e (c) e outra para a questão (d). A *trigger* das três primeiras questões deve se chamar **TR.ResultsStatus**, e a função associada **AtualizaContagem**;

A *trigger* da questão (d) deve se chamar **TR.Results** e a função relacionada **VerificaStatus**:

- (a) Ao inserir novas tuplas na tabela **Results**, incremente as quantidades inseridas nos respectivos **status** na tabela **Results.Status**. Além disso, mostre na tela a mensagem 'StatusID: <status modificado>, Contagem: <nova contagem>.'
- (b) Ao remover tuplas da tabela **Results**, diminua as quantidades removidas nos respectivos **status** da tabela **Results.Status**. Além disso, mostre na tela a mensagem 'StatusID: <status modificado>, Contagem: <nova contagem>.'
- (c) Ao atualizar tuplas da tabela **Results** (especificamente o atributo **statusid**), altere, na tabela **Results.Status** as quantidades removidas nas respectivas escuderias. Além disso, mostre na tela a mensagem 'StatusID Anterior: <status Anterior>, Contagem: <nova contagem>' e 'StatusID Atual: <status atual>, Contagem: <nova contagem>', em que a primeira mensagem é o **Status** que teve a quantidade subtraída, e a segunda se refere à contagem que teve a quantidade aumentada.

IMPORTANTE:

Vou colocar a função e os testes em partes aqui no relatório, ao passo que seria impossível tirar um print legível do código por completo.

Criação da função, gatilho e ponderações iniciais

Nesta parte, crio a função com o nome exigido pela atividade, declaro variáveis que foram necessárias durante a criação do código e, já no início do código, decidi fazer uma validação simples antes de qualquer operação, em que valido a existência do status ID. Mesmo que essa validação seja feita pela FK, criando a exceção, a exibição fica mais fácil de se entender, deixando nosso banco mais semântico.

```
-- Trigger e função para as questões a b e c;
CREATE OR REPLACE FUNCTION AtualizaContagem()
  RETURNS TRIGGER AS $$
  DECLARE
    nova_contagem INTEGER; -- Variável para armazenar a nova contagem
    nova_contagem_old INTEGER;
  BEGIN
    IF NOT EXISTS
      (SELECT 1 FROM STATUS s WHERE s.statusid = NEW.statusid)
    THEN RAISE EXCEPTION 'Status não encontrado: Impossível operar com este resultado.';
    RETURN NULL;
  END IF;
```

Abaixo, temos a trigger que foi criada, isto é, o gatilho que chamará a função que será descrita.

```
CREATE OR REPLACE TRIGGER TR_ResultsStatus  
BEFORE INSERT OR UPDATE OR DELETE ON Results  
FOR EACH ROW  
EXECUTE FUNCTION AtualizaContagem();
```

3.2.a - Inserção

3.2.a.1) Trigger e função criadas:

Código de validação

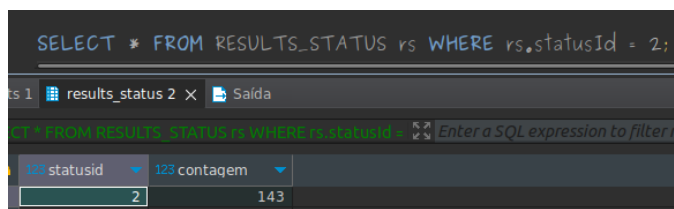
Nesta parte, lido com a inserção, utilizando a variável TG_OP para tratar separadamente o 'INSERT'. Feito isso, utilizo o SET para incrementar o valor da variável contagem, ao passo que um resultado com aquele status está sendo inserido na tabela results, e a contagem da tabela results_status deve se manter íntegra.

```
IF TG_OP = 'INSERT' THEN  
    UPDATE results_status SET contagem = contagem + 1 WHERE statusid = NEW.statusid RETURNING contagem INTO nova_contagem;  
    RAISE NOTICE 'StatusID: %, Contagem: %.', NEW.statusid, nova_contagem; -- exibindo mensagem da nova inserção  
    RETURN NEW;  
END IF;
```

3.2.a.2) Testes:

Passo 1:

Recuperando a quantidade atual de resultados que contém aquele status. Nesse caso, utilizarei o status 2, que foi selecionado aleatoriamente.



The screenshot shows a SQL query being executed in a database client. The query is: `SELECT * FROM RESULTS_STATUS rs WHERE rs.statusId = 2;`. The results are displayed in a table with two columns: `statusid` and `contagem`. The first row shows `statusid` as 2 and `contagem` as 143.

statusid	contagem
2	143

Passo 2:

Realizar uma nova inserção na tabela resultados atribuindo para o statusId o valor supracitado (2).

```
● INSERT INTO Results (ResultId, RaceId, DriverId, ConstructorId, Number, Grid, Position, PositionText, PositionOrder, Points, Laps, Time, Milliseconds, FastestLap, Rank, FastestLapTime, FastestLapSpeed, StatusId)
VALUES (999999, 18, 1, 1, 22, 1, 1, '1', 1, 10.0, 58, '1:34:50.616', 5690616, 39, 2, '1:27.452', '218.300', 2);
```

Name	Value
Updated Rows	1
Query	INSERT INTO Results (ResultId, RaceId, DriverId, ConstructorId, Number, Grid, Position, PositionText, PositionOrder, Points, Laps, Time, Milliseconds, FastestLap, Rank, FastestLapTime, FastestLapSpeed, StatusId) VALUES (999999, 18, 1, 1, 22, 1, 1, '1', 1, 10.0, 58, '1:34:50.616', 5690616, 39, 2, '1:27.452', '218.300', 2)
Start time	Mon May 29 22:41:44 BRT 2023
Finish time	Mon May 29 22:41:44 BRT 2023

Passo 3:

Validar que a mensagem exibida após a atualização foi: “StatusID: 2, Contagem: <retornada_no_passo_1 + 1>.” Além disso, recuperar a quantidade atual de resultados que contém aquele status para conferir se o valor aumentou em 1 quando comparado com o “Passo 1”.

```
SELECT * FROM RESULTS_STATUS rs WHERE rs.statusId = 2;
```

statusid	contagem
2	144

```
● INSERT INTO Results (ResultId, RaceId, DriverId, ConstructorId, Number, Grid, Position, PositionText, PositionOrder, Points, Laps, Time, Milliseconds, FastestLap, Rank, FastestLapTime, FastestLapSpeed, StatusId)
VALUES (999999, 18, 1, 1, 22, 1, 1, '1', 1, 10.0, 58, '1:34:50.616', 5690616, 39, 2, '1:27.452', '218.300', 2);
```

C
StatusID: 2, Contagem: 144.

3.2.b - Deleção

3.2.b.1) Trigger e função criadas:

Código de validação

Nesta parte, lido com a deleção, utilizando a variável TG_OP para tratar separadamente o ‘DELETE’. Feito isso, utilizo o SET para decrementar o valor da variável contagem, ao passo que um resultado com aquele status está sendo removido da tabela results, e a contagem da tabela results_status deve se manter

íntegra.

```
IF TG_OP = 'DELETE' THEN
  UPDATE results_status SET contagem = contagem - 1 WHERE statusid = OLD.statusid RETURNING contagem INTO nova_contagem;
  RAISE NOTICE 'StatusID: %, Contagem: %, OLD.statusid, nova_contagem; -- exibindo mensagem da nova deleção'
  RETURN OLD;
END IF;
```

3.2.b.2) Testes:

Passo 1:

Recuperando a quantidade atual de resultados que contém aquele status. Nesse caso, utilizarei o status 116, que foi selecionado aleatoriamente. Perceba que antes de tudo ele está com 7 na contagem.

• -- Testes para a questão b.
-- 1. Selecionando um resultado de status id aleatório para apagar um da tabela results.

```
SELECT * FROM RESULTS_STATUS rs WHERE rs.statusId = 116;
```

statusid	123 contagem
116	7

Passo 2:

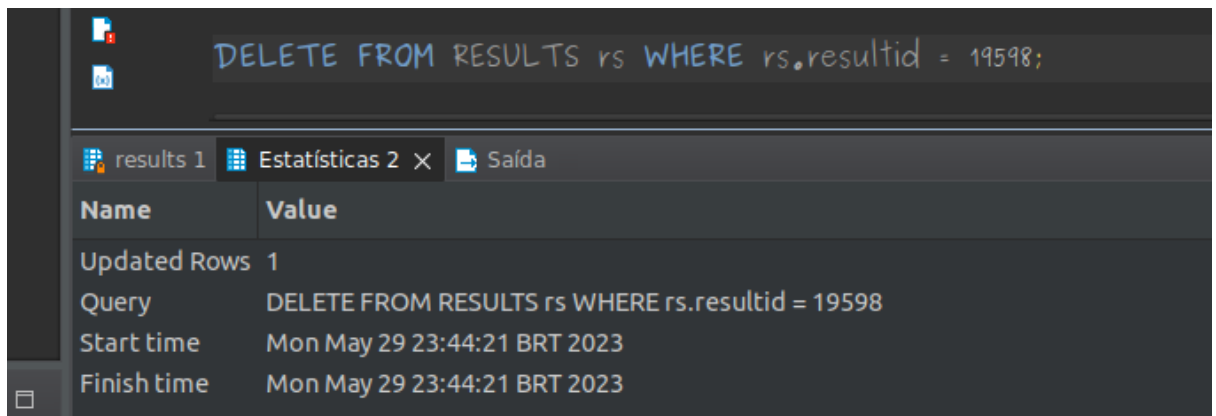
Encontrar um resultado que possui o id supracitado (116) e realizar uma deleção na tabela results.

```
SELECT * FROM RESULTS rs WHERE rs.statusId = 116;
```

	resultid	123 raceid	123 driverid	123 constructorid	123 number	123 grid	123 position	Abc position
1	17.222	710	386	66	8	8	11	11
2	17.682	731	436	172	4	19	11	11
3	19.598	815	719	133	18	18	9	9
4	19.679	818	731	110	2	12	19	19
5	19.680	818	675	110	8	28	20	20
6	19.757	821	689	87	11	13	22	22
7	20.090	835	509	160	76	28	24	24

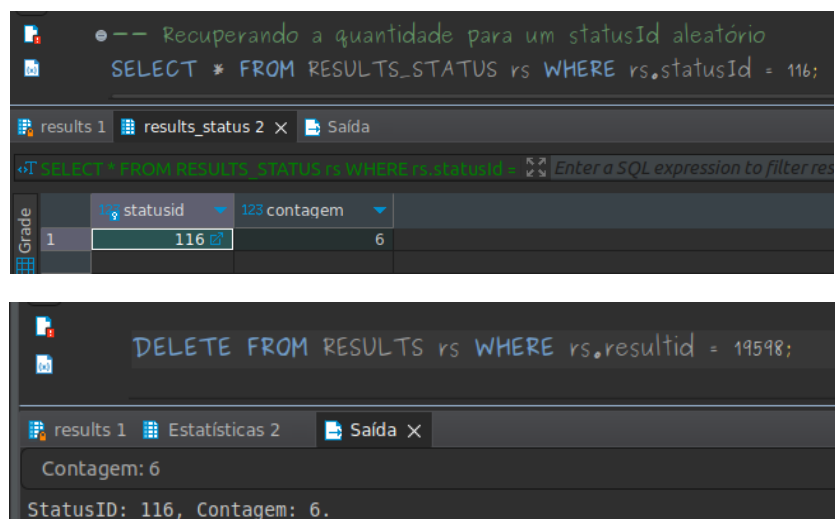
Escolhi esse terceiro que apareceu, cujo resultId é igual a 19598.

Agora, realizarei a deleção:



Passo 3:

Validar que a mensagem exibida após a atualização foi: “StatusID: 19598, Contagem: <retornada_no_passo_1 - 1>.” Além disso, recuperar a quantidade atual de resultados que contém aquele status para conferir se o valor diminuiu em 1 quando comparado com o “Passo 1”.



3.2.b - Alteração

3.2.b.1) Trigger e função criadas:

Código de validação

Nesta parte, lido com a deleção, utilizando a variável TG_OP para tratar separadamente o ‘UPDATE’. Feito isso, utilizo o SET para decrementar o valor da variável contagem para o status id antigo, utilizando o OLD.statusId, e incrementar o novo, utilizando o NEW.statusId. Com isso, consigo coordenar uma edição na tabela

results mantendo a tabela RESULTS_STATUS íntegra e consistente.

```
IF TG_OP = 'UPDATE' THEN
    IF NEW.statusid <> OLD.statusid THEN -- Esse IF me garante que o atributo statusId foi modificado
        UPDATE results_status SET contagem = contagem - 1 WHERE statusid = OLD.statusid RETURNING contagem INTO nova_contagem_old;
        UPDATE results_status SET contagem = contagem + 1 WHERE statusid = NEW.statusid RETURNING contagem INTO nova_contagem;
        RAISE NOTICE 'StatusID sendo atualizado': -- exibindo mensagem da nova edição
        RAISE NOTICE 'StatusID Anterior: x, Contagem: x', OLD.statusid, nova_contagem_old;
        RAISE NOTICE 'StatusID Atual: x, Contagem: x', NEW.statusid, nova_contagem;
        RETURN NEW;
    END IF;
END IF;
END;
## LANGUAGE plpgsql;
```

Além disso, é aqui que encerramos a função.

3.2.b.2) Testes:

Passo 1:

Para esse teste de edição, precisamos selecionar 2 status ID aleatórios, um que perderá um (-1) resultado com o determinado statusId, e outro que receberá um (+1). Para tanto, selecionei aleatoriamente os seguintes statusId.

-- 1. Selecionando 2 status ID aleatórios, um irá reduzir a contagem e o outro irá incrementar.

```
SELECT * FROM RESULTS_STATUS rs WHERE rs.statusId = 3 OR rs.statusId = 12 ;
```

statusid	contagem
3	1.046
12	1.594

Selecionei o statusId 3 para ganhar um resultado a mais e o 12 para perder um. Sendo assim, farei um update de algum registro da tabela “RESULTS” que tem o statusId como 12 mudando para 3.

Passo 2:

Encontrar um resultado que possui o id supracitado (12) e realizar uma edição na tabela “RESULTS”, alterando o statusId para 3.

-- 2. Recuperando algum resultId aleatório que contém o statusId supracitado. (Decidi trocar um 12 por um 3)

```
SELECT * FROM RESULTS rs WHERE rs.statusId = 12;
```

	resultid	123 raceid	123 driverid	123 constructorid	123 number	123 qrid	123 position	123 positiontext	123 positionorder
1	1.710	99	2	17	18	17	16	16	16
2	1.766	102	2	17	18	16	12	12	12
3	1.767	102	32	19	15	14	13	13	13
4	1.848	106	44	7	17	10	14	14	14
5	1.849	106	10	17	19	17	15	15	15
6	1.868	107	32	19	15	15	14	14	14
7	1.869	107	10	17	19	17	15	15	15
8	1.948	111	42	19	15	15	14	14	14
9	1.962	112	52	17	12	15	8	8	8
10	1.963	112	18	16	17	5	9	9	9
11	1.964	112	3	15	0	14	10	10	10

Escolhi um cujo resultId é igual a 1948.

Agora, realizarei o UPDATE:

-- 3. Realizando o update.

```
UPDATE RESULTS
SET statusId = 3
WHERE resultId = 1948;
```

results 1	Estadísticas 2	Saída
StatusID A		
StatusID Anterior: 12, Contagem: 1593		
StatusID Atual: 3, Contagem: 1047		

Passo 3:

Validar que a mensagem exibida após a atualização foi: "

"StatusID Anterior: 12, Contagem: 1593"

"StatusID Atual: 3, Contagem: 1047"

Além disso, recuperar a quantidade atual de resultados que contém aquele status para conferir se o valor do statusId 12 diminuiu em 1 e se o statusId 3 aumentou 1 quando comparado com a execução no "Passo 1".

```
SELECT * FROM RESULTS_STATUS rs WHERE rs.statusId = 3 OR rs.statusId = 12 ;
```

results 1	results_status 2	Saída
SELECT * FROM RESULTS_STATUS rs WHERE rs.statusId = 3 OR rs.statusId = 12 ;		
Grade	statusid	123 contagem
1	12	1.593
2	3	1.047

results 1

results_status 2

Saída X

StatusID A

StatusID Anterior: 12, Contagem: 1593
StatusID Atual: 3, Contagem: 1047

3.2 - Exercício 2 (d)

- (d) Antes de inserir novas tuplas na tabela **Results** ou atualizar o **Statusid** de alguma delas, verifique se o **StatusId** fica negativo. Caso fique, levante uma exceção com a mensagem 'StatusID Negativo! Operação cancelada.' e não execute a operação.

Execute comandos para testar cada um dos casos e apresente os testes e resultados no relatório.

d.1) Trigger e função criadas:

```
-- Trigger e função para questão d:
CREATE OR REPLACE FUNCTION VerificaStatus()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.statusid < 0 THEN
        RAISE NOTICE 'StatusID Negativo! Operação cancelada.'; -- exibindo mensagem de erro por status negativo.
        RETURN NULL;
    END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TR_Results
BEFORE INSERT OR UPDATE ON Results
FOR EACH ROW
EXECUTE FUNCTION VerificaStatus();
```

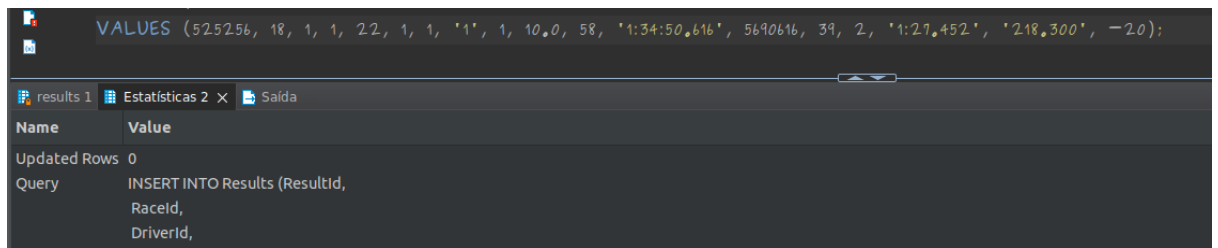
d.2) Tentando inserir um resultado com StatusId negativo:

```
INSERT INTO Results (ResultId,
    RaceId,
    DriverId,
    ConstructorId,
    Number,
    Grid,
    Position,
    PositionText,
    PositionOrder,
    Points,
    Laps,
    Time,
    Milliseconds,
    FastestLap,
    Rank,
    FastestLapTime,
    FastestLapSpeed,
    StatusId)
VALUES (1, 18, 1, 1, 22, 1, 1, '1', 1, 10.0, 58, '1:34:50.616', 5690616, 39, 2, '1:27.452', '218.300', -5);
```

results 1 Estatísticas 2 Saída X

Insira parte da mensagem a ser pesquisada aqui

StatusID Negativo! Operação cancelada.



Valido que nenhuma linha foi alterada para a tentativa de inserção com statusId -20, além da mensagem da saída:
“StatusID Negativo! Operação cancelada.