PERFORM

O comando PERFORM permite a execução de um comando SELECT desprezando o resultado do comando.

```
PERFORM query;
```

A variável especial FOUND é definida como verdadeiro se a instrução produzir pelo menos uma linha, ou falso se não produzir nenhuma linha.

PERFORM

Exemplo:

```
CREATE FUNCTION secao_funcionarios( cod secao.codigo%TYPE )
RETURNS INTEGER AS $$
  DECLARE
    ret    INTEGER = 0;
  BEGIN
    PERFORM * FROM secao WHERE codigo=cod;
    IF FOUND THEN
      SELECT INTO ret COUNT(*) FROM funcionario WHERE
funcionario.secao=cod;
    ELSE
      RAISE EXCEPTION 'Secao % nao existe', cod;
    END IF;
    RETURN ret;
  END;
$$ LANGUAGE plpgsql;
```

Teste:

```
SELECT secao_funcionarios( 'ADM' );
 secao_funcionarios
--------------------
```

EXCEPTION

A cláusula **EXCEPTION** permite capturar e recuperar erros ocorridos dentro de um bloco.

```
BEGIN
    instruções
EXCEPTION
    WHEN condição [ OR condição ... ] THEN
        instruções_do_tratador
    [ WHEN condição [ OR condição ... ] THEN
          instruções_do_tratador
      ... ]
END;
```

Caso não ocorra nenhum erro no processamento do bloco, as instruções da cláusula **EXCEPTION** são ignoradas. Mas se acontecer algum erro no processamento do bloco, o processamento das instruções é abandonado e o controle passa para a lista de **EXCEPTION**.

EXCEPTION

Quando ocorre um erro, são executadas as instruções_do_tratador da primeira condição correspondente ao erro ocorrido e depois o controle passa para a instrução seguinte ao END. Se não for encontrada nenhuma correspondência, o erro se propaga para fora do bloco como se a cláusula EXCEPTION não existisse.

Se o bloco estiver contido dentro de outro bloco contendo a cláusula EXCEPTION, o erro é tratado por esse bloco envoltório. Se não houver nenhum bloco superior com tratamento de erros, o processamento da função é interrompido.

Caso ocorra um novo erro dentro das instruções_do_tratador selecionadas, este não poderá ser capturado por esta cláusula EXCEPTION, mas é propagado para fora. Uma cláusula EXCEPTION envoltória pode capturá-lo.

EXCEPTION

A identificação das condições de erro devem ser vistas na documentação do PostgreSQL. Não há diferença entre letras maiúsculas e minúsculas nos nomes das condições.

A condição especial OTHERS corresponde a qualquer erro, exceto QUERY_CANCELED.

Quando um erro é capturado pela cláusula EXCEPTION, as variáveis locais da função PL/pgSQL permanecem como estavam quando o erro ocorreu, mas todas as modificações no estado persistente do banco de dados dentro do bloco são desfeitas.

Para relançar o erro dentro de bloco, utiliza-se o comando RAISE sem nenhum parâmetro.

EXCEPTION

Exemplo:

```
CREATE FUNCTION revenda_lucromedio( cod revenda.codigo%TYPE )
RETURNS venda.valor%TYPE AS $$
  DECLARE
    lucro       venda.valor%TYPE = 0;
    quantidade  INTEGER = 0;
    Valores     RECORD;
    ret         venda.valor%TYPE = 0;
  BEGIN
    FOR valores IN SELECT venda.valor, automovel.preco FROM venda,
automovel WHERE revenda=cod AND automovel.codigo=venda.automovel
LOOP
      quantidade = quantidade+1;
      lucro = lucro + (valores.valor-valores.preco);
    END LOOP;
    ret = COALESCE(lucro,0)/quantidade;
    RETURN ret;
    EXCEPTION
      WHEN DIVISION_BY_ZERO THEN
        RAISE NOTICE 'Nao foram encontradas vendas para revenda
%',cod;
        RETURN 0;
    END;
$$ LANGUAGE plpgsql;
```

EXCEPTION

Teste:

```
SELECT revenda_lucromedio('08');
NOTICE:  Nao foram encontradas vendas para revenda 08
 revenda_lucromedio
--------------------
                   0
```

Variáveis Especiais

Para obter informações sobre o erro ocorrido, é possível utilizar variáveis especiais que existem apenas dentro do bloco de tratamento de erro.

A variável SQLSTATE contém o código do erro ocorrido.

A variável SQLERRM contém a mensagem do erro.

Variáveis Especiais

Exemplo:

```
CREATE FUNCTION revenda_lucromedio( cod revenda.codigo%TYPE )
RETURNS venda.valor%TYPE AS $$
  DECLARE
    lucro       venda.valor%TYPE = 0;
    quantidade  INTEGER = 0;
    Valores     RECORD;
    ret         venda.valor%TYPE = 0;
  BEGIN
    FOR valores IN SELECT venda.valor, automovel.preco FROM venda,
automovel WHERE revenda=cod AND automovel.codigo=venda.automovel
LOOP
      quantidade = quantidade+1;
      lucro = lucro + (valores.valor-valores.preco);
    END LOOP;
    ret = COALESCE(lucro,0)/quantidade;
    RETURN ret;
    EXCEPTION
      WHEN OTHERS THEN
        RAISE NOTICE 'Erro (%) - %',SQLSTATE,SQLERRM;
        RETURN 0;
    END;
$$ LANGUAGE plpgsql;
```

Variáveis Especiais

Teste:

```
SELECT revenda_lucromedio('08');
NOTICE:  Erro (22012) - division by zero
 revenda_lucromedio
--------------------
                   0
```

GET STACKED DIAGNOSTICS

Outra forma de obter informações sobre o erro é utilizando o comando **GET STACKED DIAGNOSTICS**.

```
GET STACKED DIAGNOSTICS variable = item [ , ... ];
```

O valor do item deve ser atribuído a uma variável do tipo correspondente ao item. Atualmente é possível obter o valor dos seguintes itens:

RETURNED_SQLSTATE retorna um texto com o código do erro.

MESSAGE_TEXT retorna o texto da mensagem de erro.

PG_EXCEPTION_DETAIL retorna o texto do detalhamento da mensagem de erro.

PG_EXCEPTION_HINT retorna o texto da dica sobre o erro.

PG_EXCEPTION_CONTEXT retorna linha(s) de texto descrevendo o call stack.

GET STACKED DIAGNOSTICS

Exemplo:

```
CREATE FUNCTION revenda_lucromedio( cod revenda.codigo
%TYPE ) RETURNS venda.valor%TYPE AS $$
  DECLARE
    lucro          venda.valor%TYPE = 0;
    quantidade     INTEGER = 0;
    Valores        RECORD;
    ret            venda.valor%TYPE = 0;
    text_message   TEXT;
    text_sqlstate TEXT;
    text_detail    TEXT;
    text_hint      TEXT;
  BEGIN
    FOR valores IN SELECT venda.valor, automovel.preco FROM
venda, automovel WHERE revenda=cod AND
automovel.codigo=venda.automovel LOOP
      quantidade = quantidade+1;
      lucro = lucro + (valores.valor-valores.preco);
    END LOOP;
    ret = COALESCE(lucro,0)/quantidade;
    RETURN ret;
```

GET STACKED DIAGNOSTICS

```
    EXCEPTION
      WHEN OTHERS THEN
        GET STACKED DIAGNOSTICS text_message = MESSAGE_TEXT,
                    text_sqlstate = RETURNED_SQLSTATE,
                    text_detail = PG_EXCEPTION_DETAIL,
                    text_hint = PG_EXCEPTION_HINT;
        RAISE NOTICE 'Erro: (%) -
%',text_sqlstate,text_message;
        IF CHAR_LENGTH( TRIM(BOTH text_detail) )>0 THEN
          RAISE NOTICE 'Detalhamento:';
          RAISE NOTICE '  %',text_detail;
        END IF;
        IF CHAR_LENGTH( TRIM(BOTH text_detail) )>0 THEN
          RAISE NOTICE 'Dica:';
          RAISE NOTICE '  %',text_hint;
        END IF;
        RETURN 0;
    END;
$$ LANGUAGE plpgsql;
```

GET STACKED DIAGNOSTICS

Teste:

```
SELECT revenda_lucromedio('08');
NOTICE:  Erro: (22012) - division by zero
 revenda_lucromedio
--------------------
                  0
```

GET DIAGNOSTICS

O comando **GET DIAGNOSTICS** permite obter o status do resultado de um comando.

```
GET [ CURRENT ] DIAGNOSTICS variable = item [ , ... ];
```

O valor do item deve ser atribuído a uma variável do tipo correspondente ao item. Atualmente é possível obter o valor dos seguintes itens:

ROW_COUNT retorna o número de linhas processadas pelo último comando SQL.

RESULT_OID retorna o **OID** da última linha inserida pelo último comando SQL, essa variável é relevante apenas após um **INSERT** em uma tabela que tenha **OID**s.

SELECT STRICT

A cláusula STRICT do comando SELECT faz com que seja gerado um erro se o comando não retornar exatamente uma linha. Se o comando não retornar nenhuma linha, é gerado o erro NO_DATA_FOUND. Se o comando retornar mais de uma linha, é gerado o erro TOO_MANY_ROWS.

SELECT STRICT

Exemplo:

```
CREATE FUNCTION salario_por_nome( value funcionario.nome%TYPE )
RETURNS funcionario.salario%TYPE AS $$
  DECLARE
    ret funcionario.salario%TYPE = 0;
    contador INTEGER;
  BEGIN
    SELECT INTO STRICT ret salario FROM funcionario WHERE
nome=value;
    RETURN ret;
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        RAISE NOTICE 'Não há funcionario com nome %',value;
        RETURN 0;
      WHEN TOO_MANY_ROWS THEN
        GET DIAGNOSTICS contador = ROW_COUNT;
        RAISE NOTICE 'Existem % funcionarios com o nome
%',contador,value;
        RETURN 0;
    END;
$$ LANGUAGE plpgsql;
```

SELECT STRICT

Teste:

```
SELECT salario_por_nome( 'Raquel' );
 salario_por_nome
------------------
          2200.00
SELECT salario_por_nome( 'Luis' );
NOTICE:  Existem 2 funcionarios com o nome Luis
 salario_por_nome
------------------
                0
SELECT salario_por_nome( 'Marcia' );
NOTICE:  Não há funcionario com nome Marcia
 salario_por_nome
------------------
                0
```

EXECUTE

O comando **EXECUTE** permite a execução de comandos dinâmicos.

EXECUTE command-string [INTO [STRICT] target] [USING expression [, ...]];

A string **command-string** é executada e se especificada a cláusula **INTO**, o resultado é armazenado em **target**, que deve ser uma variável de registro, de linha ou uma lista de variáveis simples.

Se for especificada a cláusula **STRICT**, ocorrera um erro caso o comando não retorne exatamente uma linha.

A substituição de variáveis ou expressões pode ser realizadas durante a montagem do comando a executar. É recomendado o uso da função **quote_ident** para substituição de nomes de campos, tabelas e outros e **quote_literal** para substituição de valores para realizar uma substituição segura sem problemas com uso de aspas ou outras formas de quote. Se o valor a ser substituido pode ser nulo, deve ser utilizada **quote_nullable** no lugar de **quote_literal**.

EXECUTE

Exemplo:

```
CREATE FUNCTION atualizar_funcionario( key
funcionario.matricula%TYPE, field TEXT, value TEXT )
RETURNS VOID AS $$
  DECLARE
    line    TEXT;
  BEGIN
    line = 'UPDATE funcionario ' || ' SET ' ||
           quote_ident( field ) || ' = ' ||
           quote_nullable( value ) || ' WHERE matricula='
           || quote_literal ( key );
    EXECUTE line;
  END;
$$ LANGUAGE plpgsql;
```

Teste:

```
SELECT atualizar_funcionario('28912','nascimento',NULL);
SELECT nascimento FROM funcionario WHERE
matricula='28912';
 nascimento
------------
```

EXECUTE

A cláusula USING permite a substituição de variáveis ou expressões para valores do comando a ser executado.

EXECUTE

Exemplo:

```
CREATE FUNCTION atualizar_funcionario( key
funcionario.matricula%TYPE, field TEXT, value TEXT )
RETURNS VOID AS $$
  DECLARE
    line    TEXT;
  BEGIN
    line = 'UPDATE funcionario ' || ' SET ' ||
           quote_ident( field ) || ' = $1 ' ||
           ' WHERE matricula= $2 ';
    EXECUTE line USING value, key;
  END;
$$ LANGUAGE plpgsql;
```

Teste:

```
SELECT atualizar_funcionario( '28912', 'nome', 'Bruna' );
SELECT nome FROM funcionario WHERE matricula='28912';
      nome
----------------
 Bruna
```

EXECUTE

O comando **EXECUTE** também pode ser utilizado com **RETURN QUERY** no formato:

```
RETURN QUERY EXECUTE command-string [ USING expression [, ... ] ];
```

FOUND

A variável **FOUND** é uma variável especial com valor booleano indicando se o último comando executado afetou alguma linha.

Os comandos que afetam o valor da variável **FOUND** são:

SELECT INTO seta **FOUND** como **TRUE** se retornou o valor de alguma linha ou **FALSE** se nenhuma linha foi retornada

PERFORM seta **FOUND** como **TRUE** se alguma linha foi produzida e descartada ou **FALSE** se nenhuma linha foi produzida.

UPDATE, **INSERT** e **DELETE** setam **FOUND** como **TRUE** se ao menos uma linha foi afetada ou **FALSE** se nenhuma linha foi afetada.

FETCH seta **FOUND** como **TRUE** se retornar uma linha ou **FALSE** se não retornar nenhuma linha.

MOVE seta **FOUND** como **TRUE** se o cursor é reposicionado ou **FALSE** caso contrário.

FOUND

FOR e **FOREACH** setam FOUND como TRUE se iteragem uma ou mais vezes ou FALSE caso contrário. A variável é setada na saída do laço. Dentro do laço, a variável não é modificada pelo comando do laço mas pode ser alterada por comandos executados dentro do laço.

RETURN QUERY e **RETURN QUERY EXECUTE** setam **FOUND** como **TRUE** se a consulta retorna ao menos uma linha e **FALSE** se a consulta não retornar nenhuma linha.

Funções VARIADIC

As funções podem ter um número variável de argumentos, desde de que os argumentos opcionais sejam de um mesmo tipo. Esses argumentos serão recebidos no último parâmetro da função, que deve ser um array do mesmo tipo dos argumentos e deve ser declarado como VARIADIC.

Funções VARIADIC

Exemplo:

```
CREATE FUNCTION minimo(VARIADIC a numeric[]) RETURNS
numeric AS $$
  DECLARE
    x NUMERIC;
    ret NUMERIC;
  BEGIN
    FOR i IN 1..ARRAY_LENGTH( a, 1 ) LOOP
      x = a[i];
      IF ret IS NULL OR ( x < ret ) THEN
        ret = x;
      END IF;
    END LOOP;
    RETURN ret;
  END;
$$ LANGUAGE plpgsql;
```

Teste:

```
SELECT minimo( 3.4, 5, 2.1, 6, 1.3, 8 );
 minimo
--------
    1.3
```

Exercícios

1) Crie uma função que receba como parâmetros o nome, rg, curso e série de um aluno e inclua o aluno na turma do curso/série com o menor número de alunos, a função deve retornar a turma utilizada ou gerar um erro com uma mensagem explicativa caso já exista um aluno com o RG indicado ou não exista nenhuma turma para o curso/serie indicado.

2) Crie uma função que receba como parâmetros o codigo da conta, código do grupo, data inicial e data final e retorne uma tabela com código da conta, código do grupo, data, valor do lançamento e total acumulado até o lançamento, para os lançamentos correspondam aos parâmetros que foram informados. Os parâmetros podem ser deixados em branco ou nulos.

Exercícios

3) Crie uma função que receba como parâmetros o nome da cidade, nome do fabricante e modelo do automóvel e retorne uma tabela com as datas em que ocorreram vendas de automóveis, o total das vendas na data e o total acumulado de todas as vendas ate a data para as vendas que correspondam aos parâmetros que foram informados. Os parâmetros podem ser deixados em branco ou nulos.