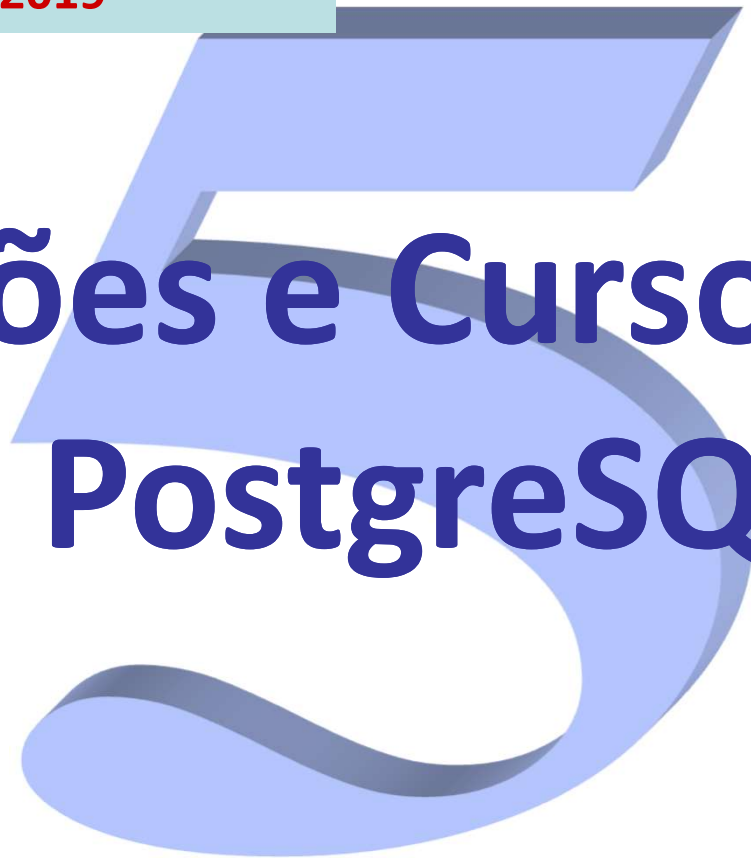
Atualizado em 24/10/2019



Exceções e Cursores no PostgreSQL

Alex Sandro
alex@ifpb.edu.br

Sumário

- *Lançando e Tratando Exceções*
  - *A instrução **raise***
  - *Bloco begin-except*
- *Cursores*
  - *Conceito*
  - *Declaração*
    - *Vinculado e não-vinculado*
  - *Abertura*
  - *Extração*
  - *Fechamento*

Exceções

- *Definição*
  - *Instrução usada para **reportar mensagens** de **erro** de volta à aplicação, ou **gerar erros** que abortarão o fluxo normal de comandos SQL*
- *A instrução **RAISE***

```
RAISE [level] <format_string>
[USING option = expressão]
  DEBUG, LOG
  NOTICE, INFO
  WARNING
  EXCEPTION  (default)
```

Literal string que especifica a mensagem
‘ …%…’, arg1, …,argn

  - *O level é opcional (default: EXCEPTION)*

Exceções

- ***Level's (report message)***
  - **INFO***:* informações implicitamente requisitadas pelo usuário
  - **NOTICE***:* informações que podem ser úteis ao usuário
  - **WARNING***:* alerta de prováveis problemas
  - **LOG***:* reportar informações de interesse do administrador
  - **DEBUG***:* fornecer informações mais detalhadas para uso pelos desenvolvedores
  - **EXCEPTION***:* abortar a execução corrente

Exceções

- ***Exemplos***

```
DO $$
BEGIN
  RAISE INFO 'MSG de informação %', current_date;
  RAISE WARNING 'MSG de alerta %', now();
  RAISE NOTICE 'MSG de notificação %', now();
  RAISE LOG 'MSG de Log: %', now();
  RAISE DEBUG 'MSG de debug %', now();
END $$;
```

  - *Apenas INFO, WARNING e NOTICE são apresentadas ao usuário* **(client_min_messages)**
  - *DEBUG e LOG ficam no log do servidor* **(log_min_messages)**
  - *EXCEPTION gera um erro.*

Parâmetros de configuração

Exceções

- ***Cláusula USING (options)***

  MESSAGE: definir texto da mensagem de erro

  HINT: fornecer mensagem de sugestão para que a causa do erro possa ser facilmente encontrada

  DETAIL: prover informação detalhada sobre o erro

```
DO $$
DECLARE
   pontos integer:= 20;
   contador integer:=1;
BEGIN
  RAISE EXCEPTION '% pontos não é suficiente para
trocar por milhas', pontos
  USING HINT = 'O mínimo é 1.000 pontos';
  RAISE NOTICE 'Tentativa %', contador;
 END $$;
```

Qual a saída do bloco de código?
Alcança o RAISE NOTICE?

Manipulando Exceções

- ***Introdução***
  - *Blocos de código ou stored procedures podem prever ou se precaver contra o lançamento de exceções*
    - *Gerado pelo **PostgreSQL** ou **intencionamente pelo usuário***
  - *Por padrão, qualquer erro ocorrido dentro de uma função PL/pgSQL aborta sua execução*
  - ***Erros** podem ser tratados internamente usando uma estrutura específica de tratamento*
    - *A lógica pode se recuperar de prováveis situações de **erro** que impediriam o funcionamento normal da rotina*

Manipulando Exceções

- ***Cenário***
  - ***Class 22: Data Exception***

```
DO $$
DECLARE
   a integer:= 5;
   b integer:= 0;
   c numeric;
BEGIN
   c := a / b;
   RAISE NOTICE 'Valor de c = %',c;
END $$;
```

```
ERROR:  division by zero
CONTEXT:  PL/pgSQL function inline_code_block line 7 at assignment
SQL state: 22012
```

Manipulando Exceções

- ***Estrutura para tratamento de erros***

```
DECLARE
  -- declarações de variáveis
BEGIN
  -- instruções PL/pgSQL
EXCEPTION
   WHEN <exceção1> [OR <exceção2>...] THEN
     -- faça algo
   [WHEN <exceção3> [OR <exceção4...] THEN
     -- faça algo ]
END $$;
```

  - *Se não ocorrer nenhum erro durante a execução do bloco, todos os comandos serão executados até o END final.*
  - *Blocos BEGIN-END podem ser acrescentados para particular o que se deseja tratar*

Manipulando Exceções

- **Tratamento**
  - **Class 22: Data Exception**

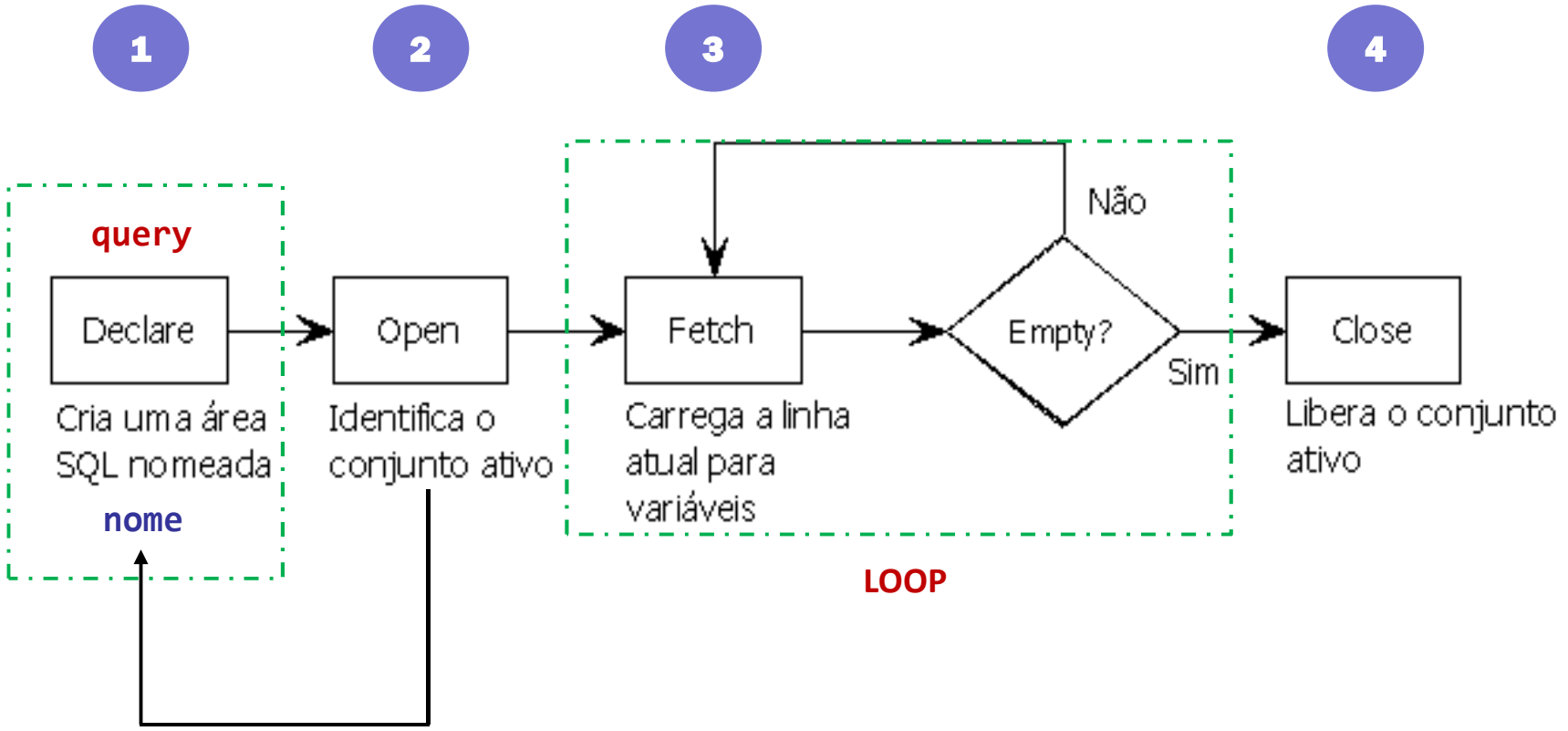
```
DO $$
DECLARE
  a integer:= 5;
  b integer:= 0;
  c numeric;
BEGIN
  c := a / b;
  RAISE NOTICE 'Valor de c = %',c;
EXCEPTION
  WHEN division_by_zero THEN
     RAISE NOTICE 'Capturada a divisão por zero!';
END $$;
```

Cursores

- ***Definição***
  - *Mecanismo que permite encapsular uma query e processor cada linha (registro) individualmente*
    - *Evita a manipulação dos registros de uma única vez*
- ***Local de uso***
  - *Triggers*
  - *Stored procedure/function*
  - *Bloco de comando*
- ***Ciclo de atividade***
  - *Seleção do **conjunto ativo,** processamento e controle linha a linha, liberação do **conjunto ativo.***

Cursores

- ***Controle de um cursor***



Tipos de Cursores

- ***Vinculados (bound)***
  - *O cursor é declarado e associado a uma query específica, que recebe argumentos ou não*
  - *Não é possível alterar a query associada ao cursor*
- ***Desvinculados (unbound)***
  - *O cursor pode ser usado para manipular qualquer query*
  - *A query é definida no momento da abertura*

Declarando um Cursor Vinculado

- ***Declaração***

```
DECLARE <nome_cursor> [[NO] SCROLL] CURSOR
[(par1 tipo, par2 tipo,...)] FOR
<instrução SELECT>;
```

  - **`scroll`**: linhas do cursor podem ser percorridas de forma não sequencial (de trás pra frente)
    - Se não for especificado, a **query** é quem determinará se ações regressivas são possíveis
    - Normalmente, se comporta como **scroll** (*default*)
  - **`par`**: argumentos cujos valores serão substituí-dos na *query* quando o cursor for aberto
    - Atentar para a compatibilidade de tipo dos argumentos no momento da associação

Declarando um Cursor Vinculado

- ***Exemplo***

```
DECLARE
   cursorFilm NO SCROLL CURSOR (ano integer)
   FOR SELECT title, release_year FROM film
   WHERE release_year = ano;
```

  - *O cursor* **cursorFilm** *resultará em um conjunto de registros contendo os campos* ***title***, ***release_year*** *de todos os filmes lançados no ano definido pelo parâmetro* ***ano***
  - *Só após o* ***OPEN*** *é que o* ***SELECT*** *será executado para navegação no conjunto ativo*

Declarando um Cursor Não-Vinculado

- ***Declaração***

```
DECLARE <nome_cursor> refcursor;
```

  - Não há referência à query

Abrindo um Cursor

- ***Abertura***
  - *O cursor **deve** ser aberto antes que possa ser usado para recuperar linhas*
  - *Para **cursor vinculado (bound)***

```
OPEN <nome_cursor> [( [arg1 :=] valor [,...])]
```

    - *Os argumentos serão substituídos na query, conforme definição na seção **DECLARE***
  - *Para **cursor desvinculado***

```
OPEN <nome_cursor> [ [NO] SCROLL] FOR query;
```

    - *Os argumentos da query (se houver) são variáveis declaradas especificadas no local oportuno*

Abrindo um Cursor

- ***Abertura***
  - *O cursor **não deve estar aberto** no momento de execução da instrução **OPEN***
  - *Uma lista de argumentos deve ser fornecida se o cursor foi declarado para receber argumentos*
  - *O cursor **executa a query e identifica o conjunto ativo**, que consiste em todas as linhas que atendem aos critérios da consulta*
  - *Uma vez selecionado o **conjunto ativo**, o apontador do cursor posiciona-se na **primeira linha** do conjunto ativo*

Abrindo um Cursor

- ***Exemplo (bound cursor)***

```
DECLARE cursorFilm NO SCROLL
CURSOR (ano integer) FOR
SELECT title, release_year FROM film WHERE
release_year = ano
OPEN cursorFilm (2006); --posicional
-- ou
OPEN cursorFilm (ano := 2006); --nomeado
```

  - ***Argumentos***
    - ***Posicional:*** *obedece a ordem de definição dos argumentos*
    - ***Nomeado****: o nome de cada argumento é especi-ficado usando* ***:=*** *para separá-lo da expressão de argumentos*

Abrindo um Cursor

- ***Exemplo (unbound cursor)***

```
-- declaração de cursor ‘não-ligado’
DECLARE
   cursorFilm refcursor;
   ano: integer;
-- abertura do cursor ‘não-ligado’
ano := 2006;
OPEN cursorFilm NO SCROLL FOR SELECT title,
release_year FROM film WHERE release_year = ano;
```

  - *Os argumentos do cursor desvinculados são definidos diretamente na query*

Abrindo um Cursor

- ***OPEN …FOR…EXECUTE (unbound)***
  - *Usado quando desejamos executar uma query dinâmica, em formato **string***
  - *A substituição dos argumentos podem ser inseridos na query dinâmica utilizando a cláusula USING*

```
OPEN <nome_cursor_unbound> [ [NO] SCROLL]
FOR EXECUTE query_string
[ USING expressão  [,...] ];
```

```
query := 'SELECT * from city WHERE country_id = $1'
key := 44;
OPEN cursorCity NO SCROLL FOR EXECUTE query
USING key;
```

Extraindo dados do Cursor

- *A instrução **Fetch***
  - ***Fetch** recupera a próxima linha do cursor e atribui a uma **variável alvo***
    - *A **variável** pode ser uma ou mais váriaveis de um tipo primitivo (separada por írgula) ou uma variável **record**.*
  - *Sintaxe*

```
FETCH [direção {FROM | IN}] <nome_cursor>
INTO <variável>;
```

    - *Certifique-se da compatibilidade dos **tipos de dados** das variáveis de acordo com o retorno do **SELECT***
    - *No caso de não haver mais linhas a recuperar, a(s) variável(is) recebe(m) **NULL***

Extraindo dados do Cursor

- *A instrução* ***Fetch***
  - *A variável especial* ***FOUND*** *pode ser utilizada para checar se a linha foi obtida ou não*

```
FETCH ...;
EXIT WHEN NOT FOUND;
```

- *A* **direção** *pode assumir um dos valores:*
  - **NEXT** *(default):* a linha da vez
  - **PRIOR***:* a linha anterior
  - **FIRST***:* a primeira linha do conjunto ativo
  - **LAST***:* a última linha do conjunto ativo
  - **ABSOLUTE count***:* a partir do início/fim do conjunto
  - **RELATIVE count***:* a partir da posição atual

Apenas para **SCROLL**

Extraindo dados do Cursor

- *A instrução* ***Fetch***
  - *Outras configurações de direcionamento:*

    FORWARD, BACKWARD (scroll)
  - *Faça o teste para verificar* ***se o cursor possui linhas*** *a serem recuperadas*
  - ***Exemplos:***

```
FETCH cursor1 INTO ano; -- NEXT (padrão)
FETCH LAST FROM cursor2 INTO var1, var2;
-- apenas para scroll cursor
FETCH RELATIVE -2 FROM cursor3 INTO var3;
FETCH ABSOLUTE 2 FROM cursor3 INTO var3;
FETCH PRIOR FROM cursor4 INTO var4,var5;
```

Movendo (posicionando) o Cursor

- *A instrução **Move***
  - *Reposiciona o "apontador" do cursor sem recuperar qualquer linha*
    - *A variável **FOUND** pode ser utilizada para checar se há linha(s) para mover*
- ***Sintaxe***

```
MOVE [direção {FROM | IN}] <nome_cursor>;
```

  - *A **direção** assume os mesmos valores e comportamento explanados no FETCH*

```
MOVE NEXT FROM cur1;
MOVE LAST FROM cur1; --scroll
MOVE RELATIVE -1 FROM cur1; --scroll
```

Extraindo dados do Cursor

- ***Exemplo***

```
DO $$
DECLARE
   cursorFilm NO SCROLL CURSOR (ano integer)
   FOR SELECT title,release_year FROM film
   WHERE release_year = ano;
   linha record;
BEGIN
   OPEN cursorFilm (2019);
   FETCH cursorFilm INTO linha; -- NEXT
   IF NOT FOUND THEN -- SEM LOOP
      RETURN;
   END IF;
END $$;
```

Fechando um Cursor

- *A instrução **CLOSE***
  - *O **CLOSE** desativa o cursor e torna o conjunto ativo indefinido*
    - *A variável (cursor) pode ser aberta novamente usando OPEN*
  - ***Sintaxe**:*

```
CLOSE <nome_cursor>;
```

  - ***Feche o cursor** sempre que terminar de processar o conjunto ativo*
  - ***Não tente** extrair dados de um cursor após ele ter sido fechado, pois será lançada uma exceção*

Extraindo dados do Cursor

- ***Exemplo COMPLETO***

```
DO $$
DECLARE
   cursorFilm NO SCROLL CURSOR (ano integer)
   FOR SELECT title, release_year FROM film
   WHERE release_year = ano;
   rec RECORD;
BEGIN
   OPEN cursorFilm (2006);
   LOOP
      FETCH cursorFilm INTO rec;
      EXIT WHEN NOT FOUND;
      RAISE NOTICE '(%) %', rec.release_year,
      rec.title;
   END LOOP;
   CLOSE cursorFilm;
END$$;
```

Cursor com OPEN e CLOSE automático(bound)

- *Uma variante do **FOR** para iterar pelas linhas retornadas por um cursor*
  - *Aplicável apenas a cursores **bound***

```
FOR <record_var> IN <bound_cursor>
[( [arg1:=]valor_argumento [,...])] LOOP
    instruções;
END LOOP;
```

  - *O **FOR** automaticamente abre o cursor e fecha quando o **LOOP** é encerrado*
  - *A lista de argumentos deve ser fornecida apenas se o cursor foi declarado para receber argumentos*
  - *Não há a necessidade de uso do **FETCH***

Cursor com OPEN e CLOSE automático(bound)

- *Exemplo*

```
DO $$
DECLARE
   cursorFilm NO SCROLL CURSOR (ano integer)
   FOR SELECT title, release_year FROM film LIMIT 10
   WHERE release_year = ano;
BEGIN
   FOR linha IN cursorFilm (2006) LOOP
      RAISE NOTICE ' (%) %', rec.release_year,
      rec.title;
   END LOOP;
   -- CLOSE cursorFilm;
END$$;
```

A variável **record_var** é automaticamente definida como **record** e existe apenas dentro do **FOR..LOOP**. Cada linha retornada a partir do cursor é diretamente atribuída a **record_var**.

Expandindo o conhecimento

- ***Como aprimorar o conhecimento do uso de cursores no PostgreSQL?***
  - ***UPDATE e DELETE WHERE CURRENT OF*** *usando o cursor para identificar a linha*
  - ***Funções*** *que retornam cursor*
- *Um código completo em que são manipulados cursors* ***bound*** *e* ***unbound***
  - *cursor_example.sql*

Cursores - Exemplo

- ***Exercício:*** *Usando um cursor, faça um bloco de código ou sp que exiba as estatísticas das mensagens enviadas por mês em 2019, obedecendo a seguinte saída:*

```
IFPB - Banco de Dados II (db myMail)
=====================================
...
Set/2005
  Alex                                5
  Edson                               4
Out/2005
  Vanessa                             30
  Cláudia                             10
...
Total Geral:                          120
```

Referências Bibliográficas

- *PL/pgSQL Errors and Messages.* http://www.postgresqltutorial.com/plpgsql-errors-messages/
- *Errors and Messagens (PostgreSQL Documentation).* https://www.postgresql.org/docs/9.3/plpgsql-errors-and-messages.html
- *PostgreSQL Error Codes.* https://www.postgresql.org/docs/9.4/errcodes-appendix.html
- *PL/pgSQL Cursor.* http://www.postgresqltutorial.com/plpgsql-cursor/
- *Cursors (PostgreSQL documentation).* https://www.postgresql.org/docs/9.4/plpgsql-cursors.html