Linguagens Procedurais no PostgreSQL: Funcoes

- Introdução à programação procedural no servidor e linguagem *plpgsql* para implementação de funções.

- ARQUITETURA/PROCESSOS
  - postmaster: (Host, único, aguardando conexões)
  - < frontend >: (Cliente, ex: psql)
  - postgres: (Servidor, um para cada cliente)

Linguagens Procedurais no PostgreSQL: Funcoes

- OBJETIVOS
  - Criar funções e gatilhos;
  - Padronizar acesso por vários programas;
  - Orientar execuções para o servidor;
  - Controlar acesso via GRANT específico;
  - Adicionar estruturas de controle;
  - Realizar calculos complexos;

Funções Implantação

- O SGBD armazena (catálogo) comandos em linguagens procedurais;
- Há um interpretador da linguagem;
- O interpretador(handler) também é uma função
- Linguagens: PL/PgSQL, PL/Tcl, PL/Perl e PL/Phyton
- Devem ser habilitadas pelo super-usuário, por exemplo:
  - CREATE LANGUAGE plpgsql template1
  - CREATE LANGUAGE plpgsql dbX
- Outros interpretadores podem ser desenvolvidos pelo usuário;

PL/PgSQL:EXEMPLO

```
CREATE OR REPLACE FUNCTION ehoras(employee.ssn%TYPE)
RETURNS DECIMAL(6,2) AS $
DECLARE
  myssn alias for $1;
  myhoras DECIMAL(6,2);
BEGIN
  SELECT SUM(hours) INTO myhoras FROM works_on
  WHERE essn=myssn;
  RETURN myhoras;
END;
$ LANGUAGE 'plpgsql';
```

- Uso da função escrita em PL/PgSQL
  - SELECT * FROM ehoras('123456789');

Funções
Sintaxe PL/PgSQL

- UM BLOCO DE SENTENÇA

  [<< label >>]

  [DECLARE declaração de variáveis]

  BEGIN

  Sentenças

  END [label];

- Observações
  - Qualquer sentença pode ser um bloco;
  - Variáveis que precedem o bloco são inicializadas a cada chamada;
  - BEGIN/END aqui não definem transações (são definas externamente);
  - tudo é convertido para minúsculas, exceto se “...”

Funções
Sintaxe PL/PgSQL

- Comentários
  - -- line comment
  - /*
    block comment
    */

Sintaxe PL/PgSQL
Declarações de Variáveis

- Os tipos são semelhantes ao SQL

  nome [CONSTANT] tipo [NOT NULL]

  [{ DEFAULT | := } expression];

- Exemplos
  - user_id INTEGER;
  - quantity NUMERIC(5);
  - url VARCHAR;
  - myrow tablename%ROWTYPE;
  - myfield tablename.fieldname%TYPE;
  - arow RECORD;
  - quantity INTEGER DEFAULT 32;
  - url varchar := ”http://mysite.com”;
  - user_id CONSTANT INTEGER := 10;

Sintaxe PL/PgSQL
Declarações de Variáveis

- Parâmetros
  - São indicados por $1, $2, etc.
  - Alias realiza a atribuição

  subtotal ALIAS FOR $1;
- Variáveis especiais
  - FOUND
    - True ou False
    - Valores
      - TRUE – Antes de uma sentença “SELECT INTO”
      - FALSE – Caso a senteça “SELECT INTO” não retorne valores

Sintaxe PL/PgSQL
Declarações de Variáveis

```
CREATE OR REPLACE FUNCTION
  doesviewexist(INTEGER) RETURNS bool AS $$
DECLARE
  key ALIAS FOR $1;
  tabledata csmaterializedviews%ROWTYPE;
BEGIN
  SELECT * INTO tabledata FROM csmaterializedviews
  WHERE sortkey=key;
  IF NOT FOUND THEN
   RETURN false;
  END IF;
  RETURN true;
END;
$$ LANGUAGE 'plpgsql';
```

Sintaxe PL/PgSQL
Estruturas de Seleção e Controle

- Encerrar uma função
  - RETURN expression; -- encerra função
- SELEÇÃO
  - IF / ELSE
- CONTROLE
  - LOOP
  - WHILE
  - FOR

Sintaxe PL/PgSQL
Estruturas de Seleção e Controle

- SELEÇÃO
  - ```
    IF (booleanExpression) THEN
    END IF;
    ```
  - ```
    IF (booleanExpression) THEN
    ELSE
    END IF;
    ```
  - ```
    IF (booleanExpression) THEN
    ELSE IF (booleanExpression)THEN
          END IF
    END IF;
    ```
  - ```
    IF (booleanExpression) THEN
    ELSIF (booleanExpression) THEN
    ELSE
    END IF;
    ```

Sintaxe PL/PgSQL
Estruturas de Seleção e Controle

- WHILE

  [<< label >>]

  WHILE expressão LOOP

  sentenças

  END LOOP;

  WHILE

- Executa enquanto
  - Até encontrar a sentença

  EXIT [label] [WHEN expression]
  - ou

  RETURN

Sintaxe PL/PgSQL
Estruturas de Seleção e Controle

- FOR

```
[<< label >>]
FOR nomevar IN [ REVERSE ] expfrom..expto LOOP
    sentenças
END LOOP;
```

Sintaxe PL/PgSQL
Exemplo

```
CREATE OR REPLACE FUNCTION my_factorial(value INTEGER) RETURNS INTEGER AS $$
DECLARE
  arg alias from $1;
BEGIN
  IF arg IS NULL OR arg < 0 THEN
       RAISE NOTICE 'Invalid Number';
       RETURN NULL;
  ELSE
      IF arg = 1 THEN
        RETURN 1;
      ELSE
        DECLARE
          next_value INTEGER;
        BEGIN
          next_value := my_factorial(arg - 1) * arg;
          RETURN next_value;
        END;
     END IF;
  END IF;
END;
```

Tratamento de Erros

- Qualquer erro ocorrido em uma função PL/pgSQL provoca sua finalização
- É possível capturar e tratar erros

```
[ <<label>> ]
[ DECLARE
declarations ]
BEGIN
statements
EXCEPTION
WHEN condition [ OR condition ... ] THEN
  handler_statements
[ WHEN condition [ OR condition ... ] THEN
  handler_statements
... ]
END;
```

Tratamento de Erros

- Caso um erro aconteça o processamento é abandonado
- A condição de erro é procurada na lista de exceções e caso seja encontrada o tratamento para aquela condição é executado.
- Exemplos de condições de erros
  - division_by_zero
  - unique_violation
  - foreign_key_violation
  - integrity_constraint_violation
  - not_null_violation
  - check_violation
  - too_many_rows
  - no_data_found
- Para capturar qualquer tipo de exceção é utilizado o termo ‘OTHERS”:

```
WHEN OTHERS THEN
    -- Tratamento genérico para qualquer tipo de erro
```

Tratamento de Erros Exemplo

```
CREATE OR REPLACE FUNCTION company.genero(essn text) RETURNS VARCHAR AS $$
DECLARE
  sretorno varchar;
  sex company.employee.sex%type;
BEGIN
  SELECT e.sex into strict sex FROM company.employee as e WHERE e.ssn = essn;
  if sex = 'F' THEN sretorno := 'FEMALE';
  else
    if sex = 'M' THEN  sretorno := 'MALE';
    else
      sretorno := 'ERRO !';
    end if;
  end if;
  return sretorno;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE EXCEPTION 'EMPLOYEE NOT FOUND';
    --RAISE NOTICE 'EMPLOYEE NOT FOUND';
END;
```

Cursores

- Um cursor contém uma query e permite obter um conjunto de dados a partir de sua execução
- Uma função pode retornar uma referência para um cursor permitindo assim que várias linhas sejam devolvidas pela função
- Utilização de cursores
  - Declarar
  - Abrir
  - Utilizar
  - Fechar

Cursores
Declaração

- Declaração de variáveis do tipo cursor

  name [[NO] SCROLL] CURSOR [(arguments)] FOR query;

  - Uma variável cursor é do tipo *refcursor*
  - SCROLL – permite navegação para linhas anteriores à linha atual
  - arguments – permite a passagem de parâmetros na criação do cursor

- Exemplos

```
DECLARE
    curs1 refcursor; -- não está ligado a nenhuma consulta
    curs2 CURSOR FOR SELECT * FROM tenk1;
    curs3 CURSOR (key integer) IS SELECT * FROM tenk1 WHERE
    unique1 = key;
```

Cursores
Abertura

- É necessário abrir o cursor antes de seu uso
- Abrir para consulta
  - OPEN unbound_cursor [[NO] SCROLL] FOR query;
    - Esta forma de abertura serve apenas para cursores que não estão ligados a nenhuma consulta
    - Neste caso a consulta deve ser sempre um SELECT
    - `OPEN curs1 FOR SELECT * FROM foo WHERE key = mykey;`
- Abrir para execução
  - OPEN unbound_cursor [[NO] SCROLL] FOR EXECUTE query_string;
    - Esta forma de abertura serve apenas para cursores que não estão ligados a nenhuma consulta
    - Permite o uso de uma consulta dinâmica
    - `OPEN curs1 FOR EXECUTE 'SELECT * FROM' || quote_ident($1);`

Cursores
Abertura

- Abertura
  - OPEN bound_cursor [(argument_values)];
    - Neste caso o cursor já possui uma query associada ao mesmo
    - É possível a passagem de parâmetros que serão utilizados na consulta
    - Neste caso não é possível o uso de consultas dinâmcias (EXECUTE)
    - `OPEN curs2;`
    - `OPEN curs3(42);`

Cursores Utilização

- Fetch
  - FETCH [direction { FROM | IN }] cursor INTO target;
  - Exemplos
    - `FETCH curs1 INTO rowvar;`
    - `FETCH curs2 INTO foo, bar, baz;`
    - `FETCH LAST FROM curs3 INTO x, y;`
    - `FETCH RELATIVE -2 FROM curs4 INTO x;`
- Move
  - MOVE [direction { FROM | IN }] cursor;
  - Exemplos
    - `MOVE curs1;`
    - `MOVE LAST FROM curs3;`
    - `MOVE RELATIVE -2 FROM curs4;`

Cursores Utilização

- UPDATE/DELETE WHERE CURRENT OF
  - UPDATE table SET ... WHERE CURRENT OF cursor;
  - DELETE FROM table WHERE CURRENT OF cursor;
  - Permite atualizar ou excluir a linha em que o cursor está posicionado
  - Exemplo
    - `UPDATE foo SET dataval = myval WHERE CURRENT OF curs1;`

Cursores
Finalização

- CLOSE
  - Finalizar o cursor e liberar os recursos antes do final da transação
  - Permite que o cursor seja reaberto
  - Exemplo
    - `CLOSE curs1;`

Cursores - Exemplo

```
CREATE OR REPLACE FUNCTION company.wreport()RETURNS character
varying AS $$
declare
  employee company.employee%rowtype;
  c CURSOR is SELECT * FROM company.employee;
Begin
  open c;
  loop
    fetch c into employee;
    EXIT WHEN NOT FOUND;
    RAISE NOTICE 'employee % ', employee.lname;
  end loop;
  CLOSE c;
  return null;
end;
$$ LANGUAGE 'plpgsql' VOLATILE
```

Retorno de um conjunto de linhas Records

```
CREATE OR REPLACE FUNCTION company.getAllEmpoyee()
RETURNS SETOF Record AS $$
DECLARE
  r record;
BEGIN
  FOR r IN SELECT fname, minit, lname FROM company.Employee
  LOOP
    RETURN NEXT r;
  END LOOP;
RETURN;
END  $$ LANGUAGE 'plpgsql' ;
```

- Chamada da função

```
select * from company.getAllEmpoyee() as (campo1 varchar(15),
campo2 character(1), campo3 varchar(15));
```

Triggers (Gatilhos)

- Comportamento que pode ser associado ao banco de dados que é executado de forma automática quando uma determinada operação é executada
- O comportamento a ser executado é definido em uma função
- Operações INSERT, UPDATE, DELETE provocam o disparo do gatilho
- Os triggers podem ser configurados para execução antes (before) ou após (after) o evento ocorrido
- Assim que o evento mapeado ocorre a função é invocada
- Um trigger pode disparar outros triggers
- Necessário cuidado a fim de evitar uma recursão infinita de triggers

Triggers (Gatilhos)

- No POSTGRES as funções invocadas pelos triggers podem ser escritas em qualquer uma das linguagens disponíveis como: PL/pgSQL, PL/Tcl, PL/Perl, PL/Python
- Para utilização dos triggers juntamente com a linguagem PL/pgSQL é necessário dois passos:
  - Definição da função com o comportamento
  - Associar Trigger à tabela
- A definição da função é semelhante um procedimento PL/pgSQL qualquer, porém existem algumas particularidades:
  - Função Não deve possuir argumentos
  - Função deve retornar um tipo **`trigger`**

Triggers
Função com Comportamento

- A função deve ser definida da seguinte forma:

```
CREATE FUNCTION emp_stamp() RETURNS trigger AS $$
```

- A função contém variáveis especiais que podem são adicionadas pelo trigger
- A função deve retornar NULL ou um RECORD/ROW com estrutura exatamente igual à da tabela que onde o trigger foi associado

Triggers
Exemplo - Função

```
CREATE FUNCTION emp_stamp() RETURNS trigger AS $emp_stamp$
BEGIN
-- Check that empname and salary are given
IF NEW.empname IS NULL THEN
RAISE EXCEPTION ’empname cannot be null’;
END IF;
IF NEW.salary IS NULL THEN
RAISE EXCEPTION ’% cannot have null salary’, NEW.empname;
END IF;
-- Who works for us when she must pay for it?
IF NEW.salary < 0 THEN
RAISE EXCEPTION ’% cannot have a negative salary’, NEW.empname;
END IF;
-- Remember who changed the payroll when
NEW.last_date := current_timestamp;
NEW.last_user := current_user;
RETURN NEW;
END;
$emp_stamp$ LANGUAGE plpgsql;
```

Triggers
Função - Variáveis

- Ao ser invocada pelo trigger a função recebe variáveis especiais que são incluídas logo em seu inicio pelo trigger
  - NEW – Tipo RECORD contendo nova linha em INSERT/UPDATE
  - OLD – Tipo RECORD contendo linha antiga em UPDATE/DELETE
  - TG NAME – O nome do gatilho
  - TG WHEN – BEFORE or AFTER
  - TG LEVEL – ROW or STATEMENT
  - TG OP – INSERT, UPDATE or DELETE
  - TG_TABLE_NAME – Tabela onde o trigger foi disparado
  - TG_TABLE_SCHEMA – Nome do esquema da tabela
  - TG_NARGS – Número de argumentos do trigger utilizados em sua definição
  - TG_ARGV[] – Array (0..n) de texto contendo os argumentos utilizados na criação do trigger

Triggers
Associação à Tabela

- A definição de um trigger e associação do mesmo à uma tabela é feita com o comando CREATE TRIGGER

  CREATE TRIGGER name { BEFORE | AFTER } { event [OR ...] }
  ON table [FOR [EACH] { ROW | STATEMENT }]
  EXECUTE PROCEDURE funcname (arguments)

  - name – nome do trrigger
  - table – nome da tabela que terá o trigger associado
  - event – INSERT, DELETE ou UPDATE
  - FOR EACH – Como será acionado
  - ROW ou STATEMENT – Indica se o trigger será disparado por linha ou por comando
  - funcname – Nome da função que será invocada pelo trigger
  - arguments – Lista de argumentos da função separada por vírgula

Triggers
Exemplo - Definição

- CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON emp

  FOR EACH ROW EXECUTE PROCEDURE emp_stamp();

- Tabela

```
CREATE TABLE emp (
empname text,
salary integer,
last_date timestamp,
last_user text
);
```

Trigger
Exemplo - Auditoria

- TABELAS

```
CREATE TABLE emp (
empname text NOT NULL,
salary integer
);

CREATE TABLE emp_audit(
operation char(1) NOT NULL,
stamp timestamp NOT NULL,
userid text NOT NULL,
empname text NOT NULL,
salary integer
);
```

Trigger
Exemplo – Auditoria (Função)

```
CREATE OR REPLACE FUNCTION process_emp_audit() RETURNS TRIGGER AS $emp_audit$
BEGIN
--
-- Create a row in emp_audit to reflect the operation performed on emp,
-- make use of the special variable TG_OP to work out the operation.
--
IF (TG_OP = 'DELETE') THEN
INSERT INTO emp_audit SELECT 'D', now(), user, OLD.*;
RETURN OLD;
ELSIF (TG_OP = 'UPDATE') THEN
INSERT INTO emp_audit SELECT 'U', now(), user, NEW.*;
RETURN NEW;
ELSIF (TG_OP = 'INSERT') THEN
INSERT INTO emp_audit SELECT 'I', now(), user, NEW.*;
RETURN NEW;
END IF;
RETURN NULL; -- result is ignored since this is an AFTER trigger
END;
$emp_audit$ LANGUAGE plpgsql;
```

Trigger
Exemplo – Auditoria (Trigger)

- Definição do Trigger

```
CREATE TRIGGER emp_audit
AFTER INSERT OR UPDATE OR DELETE ON emp
FOR EACH ROW EXECUTE PROCEDURE process_emp_audit();
```

JAVA – JDBC
Java Database Connectivity

- Permite o acesso a banco de dados
- Uma das formas de acesso é utilizando o driver JDBC-ODBC que permite a conexão através de um DRIVER OBDC
- O ODBC (Open Database Connectivity) é um padrão para acesso aos banco de dados mais utilizados no mercado: SQLSERVER; ORACLE; MYSQL; POSTGRES; MS ACCESS;
  - COMO FUNCIONA
  - TIPOS DE DRIVERS
  - COMO UTILIZAR