Câmpus
Anápolis de Ciências
Exatas e Tecnológicas
Henrique Santillo

Universidade
Estadual de Goiás

Curso: Sistemas de Informação
Disciplina : Banco de Dados II

Procedure
Triggers e Funções

Prof. M.e. Guiliano Rangel Alves

Stored Procedure PostgreSQL

- **O Que é Stored Procedure**

Um procedimento armazenado (Stored Procedure) , é uma coleção de instruções implementadas em uma linguagem suportada pelo SGBD que, uma vez armazenadas ou salvas, ficam dentro do servidor de forma pré-compilada, aguardando que um usuário do banco de dados faça sua execução

Stored Procedure PostgreSQL

- **Vantagens**

Modularidade: passamos a ter o procedimento dividido das outras partes do software, podendo alterar somente às suas operações para que se tenha as modificações por toda a aplicação;

Stored Procedure PostgreSQL

- **Vantagens**

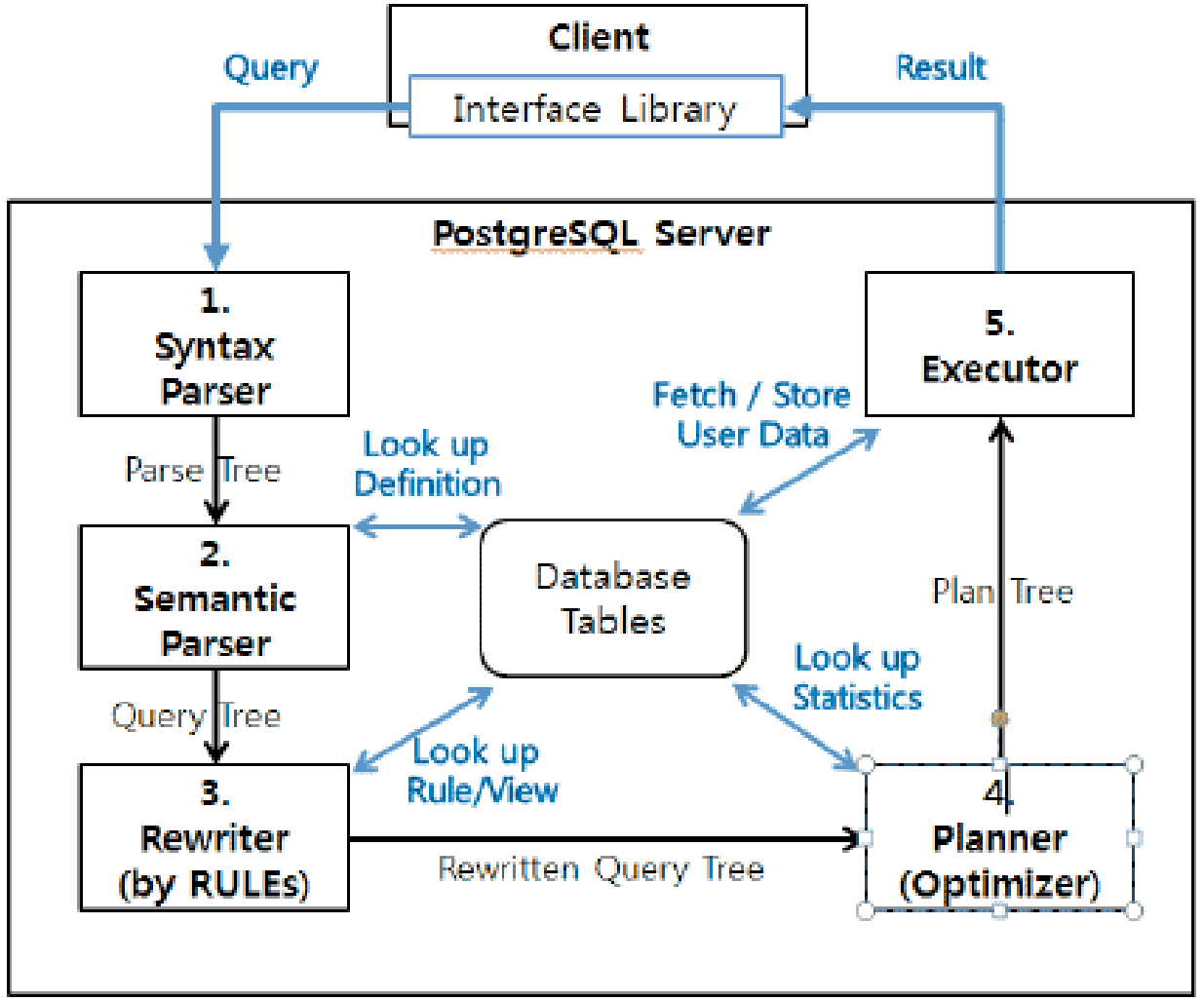
Diminuição de I/O: uma vez que é passado parâmetros para o servidor, chamando o procedimento armazenado, as operações são executadas usando processamento do servidor e no final deste, é retornado ou não os resultados de uma transação, sendo assim, não há um tráfego intenso e rotineiro de dados pela rede;

Stored Procedure PostgreSQL

- **Vantagens**

Rapidez na execução: as stored procedures, após salvos no servidor, ficam somente aguardando, já em uma posição da memória cache, serem chamados para executarem uma operação. O plano de excecução e checagem de sintaxe já foi realizada e após a primeira execução, elas se tornam ainda mais rápidas;

Stored Procedure PostgreSQL



Fonte: http://www.cubrid.org/blog/dev-platform/postgresql-at-a-glance

Stored Procedure PostgreSQL

- **Desvantagens**

Dependência : várias regras de negócio acompladas ao banco de dados, caso ocorra a necessidade de migrar a plataforma de banco de dados essas regras deverão ser reescritas na nova plataforma.

Stored Procedure PostgreSQL

- Como o PostgreSQL trata procedure
  - Os postgres não faz distinção entre function, stored procedure ou trigger como outros bancos de dados, ele trata tudo como function diferenciando um do outro apenas pelo retorno da function.
  - Suporte a várias linguagens:
    - Não procedurais: SQL
    - Procedurais: PL/pgSQL, PL/Tcl, PL/Perl, etc
    - Linguagen externa: c, c++, java, etc
  - Listagem Linguagens instaladas:
    - select * from pg_language;

Stored Procedure PostgreSQL

- **Sintaxe**

```
CREATE FUNCTION nome_funcao RETURN tipo_retorno as $conteudo$

<< comando em PL/pgSQL >>

$conteudo$ LANGUAGE plpgsql;
```

Stored Procedure PostgreSQL

- Considerações

- Em um Stored Procedure podemos referenciar tabelas, Views, outras Stored Procedures e tabelas temporárias.

Stored Procedure PostgreSQL

Exemplo:

- **Criar/alterar Stored Procedure Listar todos clientes de Anápolis**

```
CREATE OR REPLACE FUNCTION sp_clienteAnapolis()
   RETURNS SETOF cliente AS
$bloco_marcador$
   SELECT * FROM cliente WHERE cidade='Anápolis'
$bloco_marcador$
LANGUAGE 'sql';
```

- **sp_clienteAnapolis**: é o nome da procedure,
- **$bloco_marcador$** é um identificardor para indicar onde iniciou o corpo da procedure, ao final da procedure deve aparecer novamente para indicar o fim da procedure.
- **LANGUAGE 'sql'**: language é a palavra-chave para indicar qual a linguagem que foi utilizada para escrever a procedure.
- Como o retorno é do tipo SETOF a procedure deve ser tratada como uma tabela:

```
select * from sp_clienteAnapolis()
```

Stored Procedure com parâmetros:

- **Criando**

```
CREATE OR REPLACE FUNCTION sp_cliente_cidade(varchar(20))
   RETURNS SETOF cliente AS
$bloco_marcador$

SELECT * FROM cliente WHERE cidade=$1

$bloco_marcador$
LANGUAGE 'sql';
```

- **Excecutando**

- select * from sp_cliente_cidade ('Anapolis')

Stored Procedure com parâmetros:

- **Criando**

```
CREATE OR REPLACE FUNCTION sp_cliente_cidade(varchar(20))
   RETURNS SETOF cliente AS
$bloco_marcador$

SELECT * FROM cliente WHERE cidade=$1

$bloco_marcador$
LANGUAGE 'sql';
```

- **Excecutando**
- select * from sp_cliente_cidade ('Anapolis')

Stored Procedure com parâmetros:

- Stored Procedure estrutura IF … ELSE

```
IF teste_booleano THEN
      Comandos_se_verdadeiro
ELSE
      Comandos_se_false
END IF;
```

Stored Procedure com parâmetros:

- Stored Procedure estrutura IF … ELSE

```
IF teste_booleano THEN
      Comandos_se_verdadeiro
ELSE
      Comandos_se_false
END IF;
```

- Exemplo Arquivo 05-procedure-with-condicional.sql

Triguers

- Um Trigger é um bloco de comandos Transact-SQL que é automaticamente executado quando um comando INSERT , DELETE ou UPDATE for executado em uma tabela do banco de dados.
- As Triggers são usadas para realizar tarefas relacionadas com validações, restrições de acesso, rotinas de segurança e consistência de dados ; desta forma estes controles deixam de ser executados pela aplicação e passam a ser executados pelos Triggers
- No PostgreSQL é necessário criar uma função com retorno do tipo ”trigger” e depois criar a trigger em si associado a procedure ao momento de disparo(INSERT, DELETE ou UPDATE).

Triguers

- **Sintaxe – Criar Function**

- ```
  CREATE FUNCTION nome_da_trigger_function() RETURN TRIGGER
  AS $bloco$
   -- … comandos
  $boco$ LANGUAGE plpgsql;
  ```

- **Sintaxe – associar function a trigger de uma tabela:**

- ```
  CREATE TRIGGER nome_da_trigger
  BEFORE INSERT OR UPDATE ON tabela_nome
  FOR EACH ROW EXECUTE PROCEDURE nome_da_trigger_function();
  ```

- Exemplo 06-trigger_exemple.sql

Triguers

- Referência
- http://www.postgresql.org/docs/9.3/static/plpgsql-trigger.html
- http://www.postgresql.org/docs/9.3/static/plpgsql-structure.html

Visões Materializadas

- Visões materializadas são recursos introduzidos na versão 9.3 do postgresql. Enquanto visões tradicionais reexecutam uma consulta sempre que são referenciadas, visões materializadas dispensam este esforço pelos seus dados já estarem guardados desde a sua criação ou do último refresh (atualização de visão). Pode-se dizer que uma visão materializada é um objeto que contém o resultado de uma consulta, facilitando o acesso aos dados nela contidos.

- A principal justificativa para se utilizar visões materializadas é a aceleração de consultas em grandes massas de dados. É importante observar que em sistemas com pouco espaço em disco e discos lentos, visões materializadas podem ter pouco efeito ou até impacto negativo por sobrecarregar ainda mais o hardware.

- Exemplo: 08-view_materializada.sql

- **Fonte: http://postgresqlbr.blogspot.com.br/2014/02/simples-e-util-visoes-materializadas-no.html**

FIM