



Usando PostgreSQL na Regra de Negócio de um ERP

Fabiano Machado Dias
Eduardo Wolak

Regra de negócio?

São todas as regras existentes num sistema de informação, que ditam seu comportamento, suas restrições e validações. *wikipedia*

Porque usar a regra no banco de dados PostgreSQL?

- Segurança, desempenho, escalabilidade, integridade e todas as outras características que o PostgreSQL oferece.
- Experiência em sistemas desenvolvidos em outros modelos, onde a regra de negócio ficava na aplicação ou em framework intermediário.

Problemas comuns no desenvolvimento de sistemas de gestão:

- Campos obrigatórios,
- Parametrizações,
- Funções personalizadas por cliente,
- Padronização de campos em tabelas,
- Padronização de escrita SQL, entre outros...

Objetivo:

Mostrar como resolvemos esses e outros problemas usando o PostgreSQL.

Nível de estrutura (Empresa / Filial / Unidade)

Sistemas ERP geralmente são concebidos para controlar os processos não somente de uma empresa mas também de várias filiais e unidades de negócio distintas.

Existem várias maneiras de realizar este tipo de controle, uma das mais utilizadas é separar por schemas as estruturas de cada empresa/filial/unidade.

No entanto optamos por utilizar um campo padrão UK, onde é definido a estrutura da empresa.

Exemplo:

```
CREATE TABLE produtoservico  
(pkprodutoservico serial NOT NULL, -- Campo PK  
uk integer[] NOT NULL,
```

-- Campo UK é um array que contém a regra de estrutura:

[1,2,3] - fkempresa, fkfilial, fkunidade -- Campo uk para montagem de códigos não duplos.

Caso a regra de estrutura da tabela de produto/serviço fosse estipulada a nível de EMPRESA e FILIAL teríamos a montagem do ARRAY para verificação de códigos UKs dessa maneira:

-- CONSTRAINT ukprodutoservico UNIQUE (uk, codigo)

Ou seja

UK[1,2,0] + codigo(100) -- Empresa 1, Filial 2, Unidade 0, Código do produto 100

UK[1,3,0] + codigo(100) -- Empresa 1, Filial 3, Unidade 0, Código do produto 100

(Possível, pois a regra de estrutura é Empresa e Filial)

fkempresa integer NOT NULL, -- Fk da empresa

fkfilial integer NOT NULL, -- Fk da filial

fkunidade integer NOT NULL, -- Fk da unidade

codigo character varying(20), -- Código do Produto

Relacionamentos

Somente são usados como chave única, não é usado PK composta.

Relacionamento de arquivos pais não fazem parte da PK são apenas relacionadas.

Todas as PKs são do tipo SERIAL.

Campos Obrigatórios

Utilizamos duas tabelas para controlar campos obrigatórios, onde uma armazena o nome da tabela utilizada, o campo e a mensagem de retorno para o usuário, na segunda é armazenado o nome do campo de tela e o respectivo formulário para posicionar sobre o campo que foi advertido.

A principal vantagem desse tipo de implementação é a possibilidade de criar validações em tempo de execução sem a necessidade de programação para esse tipo de controle.

Exemplo:

```
CREATE TABLE mandatory (  
  pkmandatory serial NOT NULL,  
  tabela character varying,  
  campo character varying,  
  mensagem character varying,  
  CONSTRAINT pkmandatory PRIMARY KEY (pkmandatory),  
  CONSTRAINT ukmandatory UNIQUE (tabela, campo)  
) WITH (OIDS=TRUE);
```

```
CREATE TABLE mandatory_controle (  
  pkmandatory_controle serial NOT NULL,  
  fkmandatory integer NOT NULL,  
  controle character varying,  
  tela character varying,  
  CONSTRAINT pkmandatory_controle PRIMARY KEY (pkmandatory_controle),  
  CONSTRAINT fkmandatoty FOREIGN KEY (fkmandatory)  
    REFERENCES mandatory (pkmandatory) MATCH SIMPLE  
    ON UPDATE CASCADE ON DELETE CASCADE  
) WITH (OIDS=TRUE);
```

Demonstração 1

Escrita de SQL – DML

No desenvolvimento de um sistema a escrita de comandos SELECT, INSERT, UPDATE e DELETE são extremamente importantes.

Para facilitar o desenvolvimento e padronizar esse tipo de escrita criamos uma PL que se encarrega de controlar os comandos DML do sistema.

Exemplo:

- PL Manutenção:

É chamada dentro da linguagem e se encarrega de chamar outras PLs complementares, os campos e conteúdos informados pelo usuários são passados como parâmetro dentro de um ARRAY [campo,conteúdo,campo,conteúdo] sempre no formato VARCHAR.

- PL MontaQuery:

Escreve o comando DML que foi passado por parâmetro pela PL manutenção, pesquisa no catálogo do banco o tipo de dado da coluna e monta a instrução corretamente.

- PL VerificaMandatory:

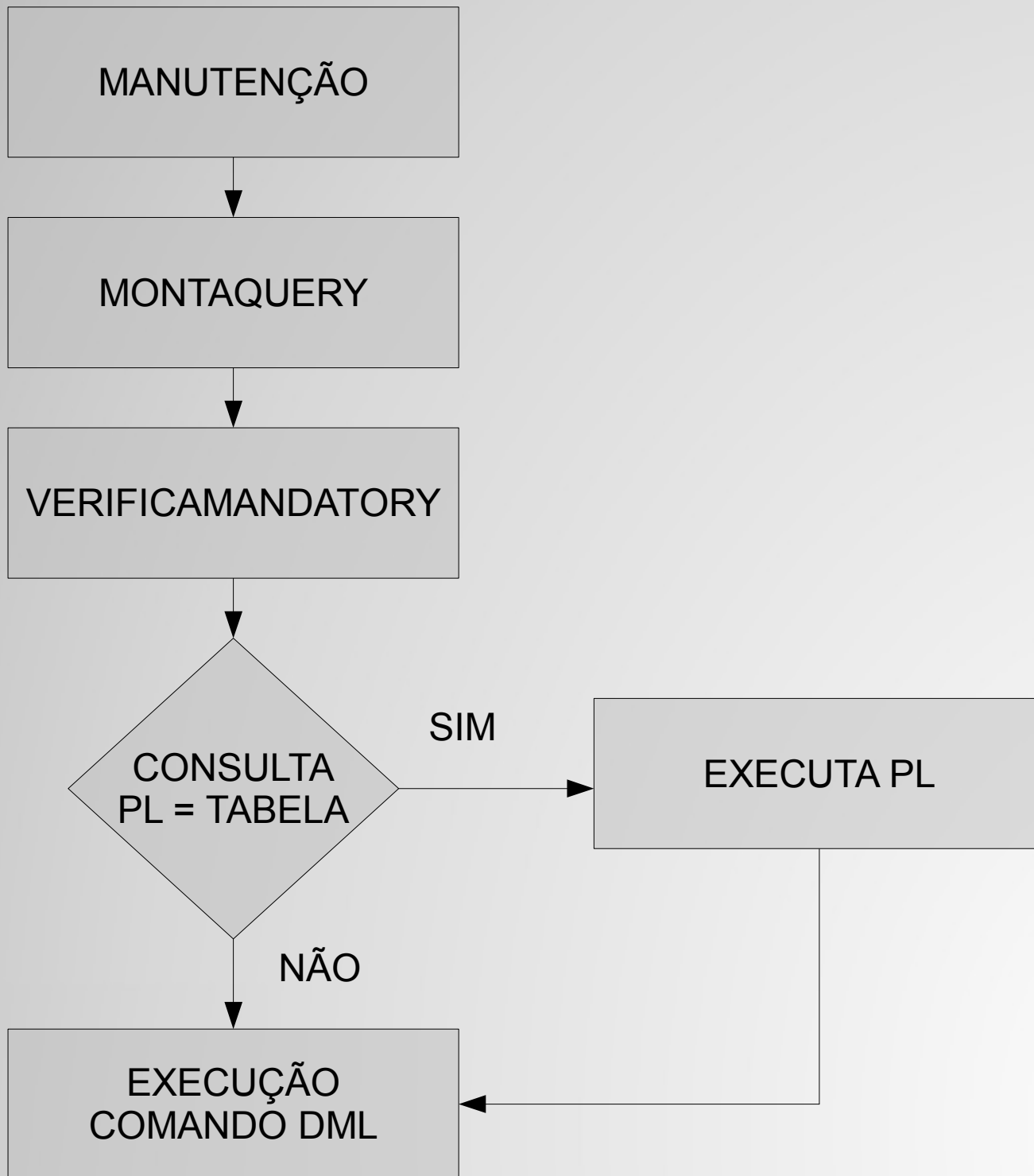
Se encarrega de verificar os campos obrigatórios do sistema.

- PL de Controle de Processamento:

Durante a execução se a PL Manutenção encontrar uma outra PL cadastrada no banco dados que corresponda ao nome da tabela a mesma é executada.

Ex.: Processos de movimentação de estoque, títulos, lançamentos contábeis.

Qualquer erro durante a execução de uma das PLs é disparado um RAISE EXCEPTION e o comando não é executado.



Demonstração 2

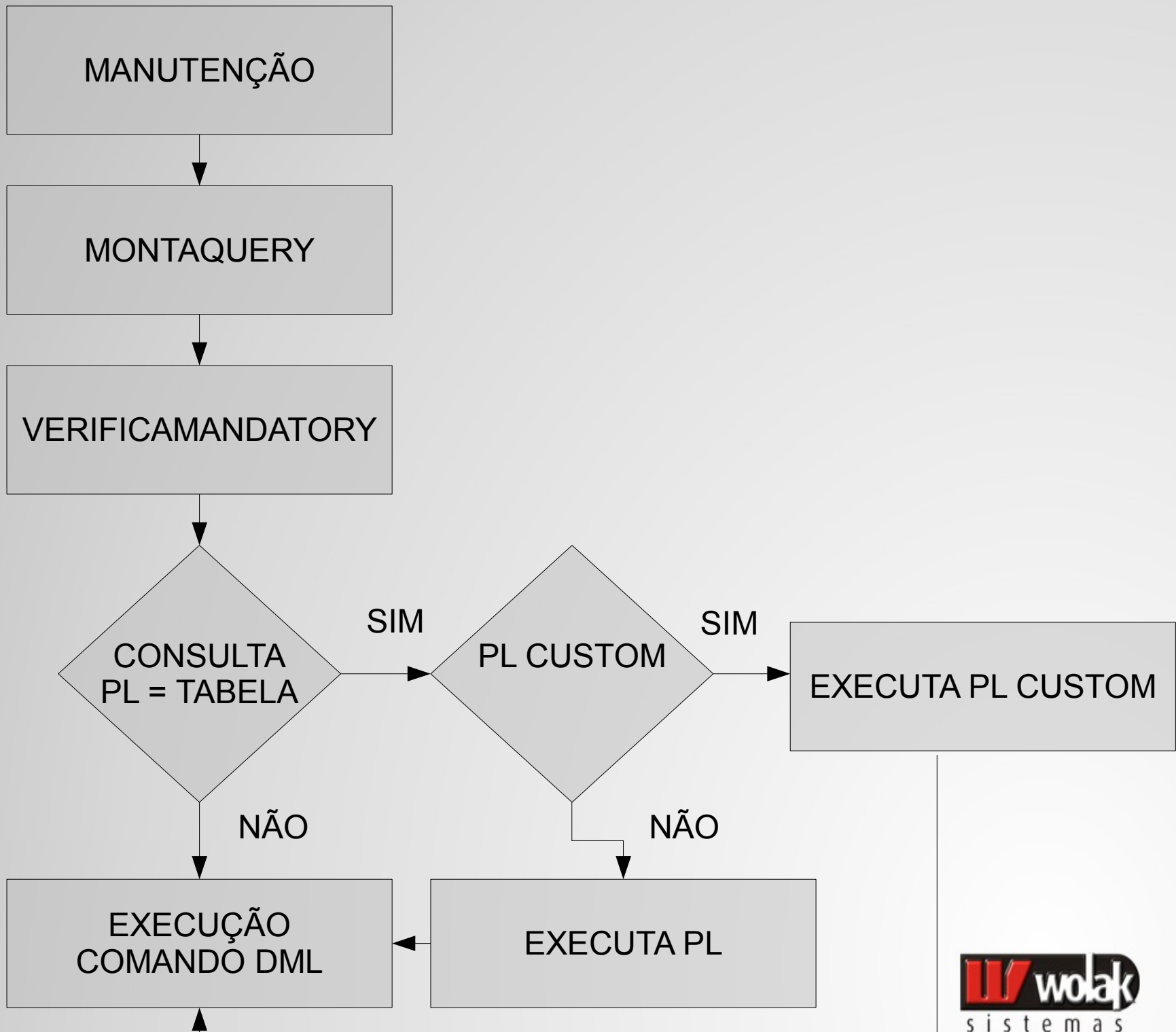
Funções Personalizadas

Em diversos casos precisamos criar personalizações para um determinado cliente. Para resolver esse tipo de problema criamos uma solução dentro da PL Manutenção onde é possível alterar o comportamento padrão do sistema.

Quando uma PL padrão é alterada, automaticamente é feita uma cópia em outro schema do banco de dados chamado custom.

Quando uma função é encontrada no schema custom o sistema passar a utilizá-la e desconsidera a função padrão.

Exemplos de utilização: Formação de preço de venda, custos, comissões de vendedores.



Demonstração 3

Cálculos

Todos os cálculos do sistema, desde os mais simples (quantidade x preço) até os mais complexos são realizados no PostgreSQL.

As PLs de controle de tabela como Nota Fiscal de Entrada realizam o cálculo e retornam o valor diretamente no campo, a classe da linguagem verifica o retorno traz na tela do usuário.

Cursors

O sistema trabalha com visualização de registro através de grids, porém um problema deste tipo de implementação é carregar uma grande quantidade de linhas.

Para resolver isto utilizamos cursores, onde a carga de dados é feita conforme o número de colunas do grid.

Um detalhe do PostgreSQL é que a atualização dos dados dos cursores só é feita quando o mesmo é fechado e aberto novamente, então a cada paginação do grid o cursor é fechado, aberto e é feito um fetch no mesmo.

Apesar disto a velocidade é excelente.

Parâmetros por tabela e/ou registro

Um dos maiores problemas de um sistema de gestão é a parametrização.

Resolvemos utilizar uma tabela onde os parâmetros são cadastrados tanto a nível de tabela e/ou nível de registro.

Exemplo de parâmetro por registro:

- Configurações de CNAB por Banco

Exemplo de parâmetro por tabela:

- Impressora de Nota Fiscal de Saída

Outro uso desta implementação é a criação de campos específicos apenas para um cliente.

Ao invés de criar um novo campo no banco de dados é possível criar um novo parâmetro e o mesmo pode ser usado como um campo de cadastro.

Demonstração 4

Campos Padrão

Em um sistema de gestão vários campos acabam sendo repetidos nas tabelas, como por exemplo:

Data de Inclusão,
Data de Alteração,
Usuário de Inclusão,
Usuário de Alteração,
Observações entre outros...

Ao invés de criamos estes campo tabela por tabela, resolvemos encapsular uma tabela dentro de outra.

Exemplo – Tabela Padrão

```
CREATE TABLE ws_controlepadrao  
(  
  fkusuarioregistrobloqueado integer,  
  ativoinativo smallint DEFAULT 1, -- Tamanho(1) 1-Ativo / 2-Inativo  
  datainativo timestamp without time zone,  
  fkusuarioinclusao integer,  
  fkusuarioalteracao integer,  
  datainclusao timestamp without time zone DEFAULT now(),  
  dataalteracao timestamp without time zone,  
  observacao character varying,  
  fkusuariomarcador integer,  
  confirmainclusao integer  
) WITH ( OIDS=TRUE);
```

Exemplo - Definição de Tabela

```
CREATE TABLE banco (  
  pkbanco serial NOT NULL,  
  uk integer[] NOT NULL,  
  fkempresa integer NOT NULL,  
  fkfilial integer NOT NULL,  
  fkunidade integer NOT NULL,  
  controle ws_controlepadrao,  
  codigo character varying(20),  
  descricao character varying(100),  
  CONSTRAINT pkbanco PRIMARY KEY (pkbanco),  
  CONSTRAINT fkempresa FOREIGN KEY (fkempresa)  
    REFERENCES cadastro (pkcadastro) MATCH SIMPLE  
    ON UPDATE CASCADE ON DELETE RESTRICT,  
  CONSTRAINT fkfilial FOREIGN KEY (fkfilial)  
    REFERENCES cadastro (pkcadastro) MATCH SIMPLE  
    ON UPDATE CASCADE ON DELETE RESTRICT,  
  CONSTRAINT fkunidade FOREIGN KEY (fkunidade)  
    REFERENCES unidade (pkunidade) MATCH SIMPLE  
    ON UPDATE CASCADE ON DELETE RESTRICT,  
  CONSTRAINT ukbanco UNIQUE (uk, codigo)  
) WITH (OIDS=TRUE);
```

Demonstração 5

A principal vantagem deste tipo de implementação é que qualquer campo adicionado na padrão ws_controlepadrão fica disponível em todas as tabelas do banco que já estejam com a ws_controlepadrão em sua definição.

Outras definições e características do projeto

- Linguagem Front-end Windev;
- Relatórios em Crystal Reports;
- Pouco uso de trigger no sistema;
- Velocidade no desenvolvimento;
- Facilidade de manutenção;
- Servidores Linux/Debian;
- Logs em discos ou partições separados;
- Sistema de arquivos XFS (noatime);
- PostgreSQL compilado.

Conclusão

Apesar de um maior trabalho no início do projeto devido ao grande número de definições e padronizações, hoje temos um desenvolvimento muito rápido devido a essa automação de tarefas rotineiras que qualquer sistema exige.

Hoje o PostgreSQL é o responsável não só pelo armazenamento dos dados mas sim de todo o processamento do sistema. O resultado ficou acima do esperado e as possibilidades são imensas.

Graças a esta escolha conseguimos desenvolver um sistema robusto, confiável, rápido e seguro.

Muito Obrigado!

Fabiano Machado Dias
fabiano@wolaksistemas.com.br

Eduardo Wolak
wolak@wolaksistemas.com.br

