

UNIVERSIDADE ESTADUAL DE MARINGÁ  
CENTRO DE TECNOLOGIA  
DEPARTAMENTO DE INFORMÁTICA  
CURSO DE ESPECIALIZAÇÃO DESENVOLVIMENTO DE SISTEMAS  
PARA WEB

**DESENVOLVIMENTO DO MÓDULO DE INVENTÁRIO COMO PARTE  
INTEGRANTE DE UM SISTEMA DE GESTÃO DE SAÚDE PÚBLICA**

CAUÊ FERRO GONZALEZ

PROF. DR. EDSON A. OLIVEIRA JUNIOR  
**ORIENTADOR**

MARINGÁ-PR  
2011

**CAUÊ FERRO GONZALEZ**

**DESENVOLVIMENTO DO MÓDULO DE INVENTÁRIO COMO PARTE  
INTEGRANTE DE UM SISTEMA DE GESTÃO DE SAÚDE PÚBLICA**

Banca Examinadora:

---

Orientador: Prof. Dr. Edson A. Oliveira Junior

---

Profa. Dra. Sandra Ferrari

---

Prof. Me. Osvaldo Alves dos Santos

*“É preciso que eu suporte duas ou três larvas se quiser conhecer as borboletas.”*

*Antoine de Saint-Exupéry*

## RESUMO

Pensando na importância de se controlar um estoque corretamente, foi desenvolvido um módulo do sistema iBsaúde capaz de automatizar a realização de um inventário periódico de produtos que atendesse a necessidade de controlar o lote e o prazo de validade dos mesmos. O desenvolvimento desse módulo visa, acima de tudo, economia de tempo, redução de esforço e de custos para o usuário final, propiciando confiabilidade e eficácia. Este trabalho apresenta a criação de tal módulo, utilizando as tecnologias PHP, *JavaScript* e PL/pgSQL.

**Palavras-chave:** PHP, *JavaScript*, PL/pgSQL, *Trigger*, Sistema Web.

## LISTA DE FIGURAS

|   |    |
|---|----|
| Figura 1 - Exemplo de código PHP/FI (PHP, 2011). .....  | 14 |
| Figura 2 - Funcionamento do PHP .....   | 16 |
| Figura 3 - Sintaxe da criação de um trigger genérico (PostgreSQL, 2009). .....                        | 24 |
| Figura 4 - Exemplo de criação de trigger (PostgreSQL, 2009). .....                                    | 24 |
| Figura 5 - Exemplo de função de trigger (PostgreSQL, 2009).....                                       | 24 |
| Figura 6 - Módulo básico de Inventário .....  | 26 |
| Figura 7 - Interface do novo módulo de inventário .....   | 27 |
| Figura 8 - Funcionamento do módulo de inventário .....  | 28 |
| Figura 9 - Criação do inventário .....  | 29 |
| Figura 10 - Exemplo de inventário de produtos que não necessitam do controle por lote e validade..... | 30 |
| Figura 11 - Tela de Geração da Movimentação .....   | 31 |
| Figura 12 - Mensagem retornada ao usuário no botão Gerar Movimentação .....                           | 31 |

## LISTA DE ABREVIATURAS

CSS – *Cascading Style Sheet*

HTML – *HyperText Markup Language*

HTTP – *HyperText Transfer Protocol*

PHP – *PHP Hypertext Preprocessor*

PL/pgSQL – *Procedural Language/PostgreSQL Structured Query Language*

SGBD – *Sistema Gerenciador de Banco de Dados*

SQL – *Structured Query Language*

# SUMÁRIO

|   |           |
|---|-----------|
| <b>INTRODUÇÃO</b> .....   | <b>8</b>  |
| 1.1 <b>CONTEXTUALIZAÇÃO</b> .....   | 8         |
| 1.2 <b>MOTIVAÇÃO</b> .....  | 8         |
| 1.3 <b>OBJETIVOS GERAIS</b> .....   | 9         |
| 1.4 <b>OBJETIVOS ESPECÍFICOS</b> .....  | 9         |
| 1.5 <b>METODOLOGIA DE DESENVOLVIMENTO</b> .....   | 10        |
| 1.6 <b>ESTRUTURA DA MONOGRAFIA</b> .....  | 10        |
| <b>2    FUNDAMENTAÇÃO TEÓRICA</b> .....   | <b>12</b> |
| 2.1 <b>O INVENTÁRIO</b> .....   | 12        |
| 2.2 <b>DESENVOLVIMENTO DE SOFTWARE PARA A PLATAFORMA WEB</b> .....  | 13        |
| 2.3 <b>DESENVOLVIMENTO PARA A PLATAFORMA WEB USANDO A TECNOLOGIA PHP</b> .....  | 13        |
| 2.3.1 <i>Utilizando o PHP</i> .....   | 15        |
| 2.3.2 <i>Validação Client-Side com JavaScript</i> .....   | 16        |
| 2.3.3 <i>Utilizando trigger para atualizar o estoque</i> .....  | 21        |
| <b>3    DESENVOLVIMENTO DO MÓDULO DE INVENTÁRIO IBISAÚDE</b> .....  | <b>26</b> |
| 3.1 <b>VISÃO GERAL DO MÓDULO DE INVENTÁRIO BÁSICO</b> .....   | 26        |
| 3.2 <b>VISÃO GERAL DO MÓDULO DE INVENTÁRIO COM CONTROLE DE LOTE E VALIDADE</b> .....                                  | 27        |
| 3.3 <b>PRINCIPAIS CARACTERÍSTICAS DO MÓDULO DE INVENTÁRIO</b> .....   | 28        |
| 3.3.1 <i>Possibilidade de cadastrar diversos lotes e validades de cada produto</i> .....                              | 28        |
| 3.3.2 <i>Diferenciação entre produtos com as informações de lote e validade e produtos sem tais informações</i> ..... | 30        |
| 3.3.3 <i>Geração da movimentação (Entrada ou Saída) para atualizar o estoque</i> .....                                | 30        |
| <b>4    CONCLUSÕES E TRABALHOS FUTUROS</b> .....  | <b>32</b> |
| <b>REFERÊNCIAS BIBLIOGRÁFICAS</b> .....   | <b>33</b> |

## INTRODUÇÃO

### 1.1 Contextualização

Segundo o dicionário Priberam da Língua Portuguesa<sup>1</sup>, a palavra inventário vem do latim *inventarium* e significa uma relação de bens, acompanhada por um documento ou lista, informando onde se encontram os mesmos. Pode conter ou não uma descrição detalhada ou minuciosa dos itens em questão.

A iBltech<sup>2</sup> é uma empresa especializada em software de gestão de saúde pública, tendo apenas prefeituras como clientes, mais especificamente secretarias de saúde.

Em um dos clientes da iBltech, existe um almoxarifado, cujo estoque de produtos (materiais e medicamentos) utilizados pela secretaria e postos de saúde deve ser controlado. Quando um carregamento chega, os produtos são cadastrados em um sistema de gestão de saúde pública, o iBlsaúde, na forma de entrada. Quando um produto é enviado para um posto de saúde, farmácia ou para a própria secretaria de saúde, é gerado um movimento de saída no sistema. O estoque é calculado de acordo com os valores das entradas e saídas geradas pelo sistema.

Pelo menos uma vez por ano, o almoxarifado suspende suas atividades para realizar o inventário, contando todos os produtos do estoque e confrontando com os dados apontados pelo sistema, com o intuito de manter a consistência dos dados persistentes.

### 1.2 Motivação

O iBlsaúde possui atualmente um gerenciador de inventário que não observa a existência de lotes e as respectivas validades dos produtos. Por muito tempo, o controle foi realizado apenas pelo nome do produto e pela quantidade. Ao longo dos anos, a importância de um controle minucioso foi tornando-se evidente. Porém, por causa da urgência e da demanda em atender às requisições dos produtos, tal controle vem sendo realizado manualmente.

No globalizado mundo atual e tendo em vista que o ser humano é bem suscetível ao erro, considera-se praticamente inadmissível que uma atividade tão

---

<sup>1</sup> <http://www.priberam.pt/dlpo/Default.aspx>

<sup>2</sup> <http://www.ibitech.com.br>

importante quanto o controle efetivo de estoque, de qualquer segmento de mercado, seja realizada manualmente.

Com base nisso, um novo sistema de inventário foi proposto aos clientes, o qual engloba o gerenciamento de estoque de qualquer tipo de produto, possuindo ou não lote e validade.

O projeto foi imediatamente aprovado pelos clientes, uma vez que abrange todas as novas funcionalidades necessárias.

### **1.3 Objetivos Gerais**

Este trabalho teve como objetivo geral desenvolver um módulo automatizado para controlar efetivamente o estoque de produtos do almoxarifado, por meio de um inventário realizado periodicamente. Tal inventário aponta o lote e a validade dos produtos gerenciados, dentre outras funcionalidades.

### **1.4 Objetivos Específicos**

Este trabalho teve como objetivos específicos:

- Analisar o problema apresentado pelos clientes;
- Realizar o planejamento das atividades a serem desenvolvidas;
- Desenvolver o protótipo do sistema;
- Apresentar o protótipo;
- Desenvolver o sistema;
- Testar o sistema; e
- Apresentar o produto final.

## 1.5 Metodologia de Desenvolvimento

Quando o cliente apresentou o problema, foi agendada uma reunião para o levantamento dos requisitos do sistema com o intuito de iniciar o seu planejamento. A fase de analisar o problema apresentado foi bem sucinta, mantendo o foco nos itens mais importantes dentre os requisitos levantados.

Foi na fase de planejamento que os detalhes foram identificados e analisados para que o desenvolvimento de tal fase fosse bem sucedido. Esse planejamento envolveu um amplo estudo do sistema iBlsaúde, para levantar os pontos que seriam afetados com o desenvolvimento do novo módulo de inventário.

Durante o desenvolvimento do protótipo do sistema de inventário, foram decididas quais tecnologias deveriam ser adotadas. Apesar de a tecnologia base já estar definida, uma vez que o sistema todo é desenvolvido em PHP, foi necessário implementar alguns recursos utilizando outras tecnologias como *JavaScript* e *PL/pgSQL*. No momento de apresentar o protótipo desenvolvido, algumas tomadas de decisão foram necessárias por causa de alguns detalhes não mencionados no levantamento de requisitos.

O desenvolvimento do sistema foi a fase mais dispendiosa. Precisou ser interrompida por alguns dias para estudos relacionados como, por exemplo, *triggers*. Tal estudo foi de extrema importância já que a nova solução exige que o estoque esteja sempre atualizado, a cada movimentação, em vez de ser calculado com base em entradas e saídas, como era feito anteriormente.

A fase de testes aconteceu simultaneamente à fase de desenvolvimento e se estendeu por mais algumas semanas para reduzir o número de falhas identificadas após a implantação.

No momento de apresentar a solução final, o cliente já havia testado o sistema em uma base de dados de testes e aprovado o resultado.

## 1.6 Estrutura da Monografia

Este trabalho está organizado da seguinte forma: o Capítulo 2 apresenta uma visão geral das tecnologias utilizadas para desenvolver o módulo de inventário, bem

como uma abordagem detalhada do processo de funcionamento de um inventário; o Capítulo 3 apresenta o novo sistema de inventário e as suas principais características, telas e modelagem; e o Capítulo 4 apresenta as conclusões e sugestões de trabalhos futuros.

## 2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos básicos para o entendimento de como um inventário deve funcionar e contextualiza as tecnologias utilizadas no desenvolvimento do inventário do sistema iBIsaúde.

### 2.1 O Inventário

O inventário dos produtos está intimamente ligado ao controle de estoque. De acordo com o portal de contabilidade, coordenado por Américo Garcia Parada Filho<sup>3</sup>, o controle de estoque realizado diariamente, por meio dos registros de entrada e saída de produtos, é conhecido como inventário perpétuo, e o inventário realizado periodicamente é chamado de inventário periódico.

O inventário periódico, foco deste trabalho, tem por principal função comparar as quantidades dos itens no estoque com as quantidades movimentadas e computadas no sistema, fazendo assim uma avaliação e, possivelmente, uma correção do saldo físico dos itens em determinados locais do estoque (FILHO, 2011).

Nas movimentações do dia a dia, podem ocorrer algumas variações de quantidade, devido a erro humano ou mesmo à perda ou dano do produto durante o transporte, por exemplo. Essas variações sofridas pelo inventário perpétuo geram inconsistências no estoque, que devem ser corrigidas em algum momento. Esse momento é quando se realiza o inventário periódico, tornando o estoque informado pelo sistema correspondente à realidade.

O controle de estoque é realizado com base nas movimentações de entradas e saídas de produtos. Quando um produto é recebido pelo centro estocador, é gerado um movimento de entrada no sistema. Essa entrada é imediatamente persistida no estoque, atualizando sua quantidade. O mesmo ocorre quando um produto será consumido. Um movimento de saída é gerado e, conseqüentemente, a quantidade no estoque é atualizada.

---

<sup>3</sup> <http://www.cosif.com.br/mostra.asp?arquivo=contabil04inventario#inventario>

## 2.2 Desenvolvimento de Software para a Plataforma Web

O mundo tecnológico evolui com uma velocidade muito grande, e onde antes as aplicações eram locais e simples, hoje são cada vez mais complexas e distribuídas. Dessa forma, a maneira de desenvolver sistemas sofre importantes alterações ao longo do tempo, que acabam por se mostrar grandes mudanças, tanto de paradigmas quanto de práticas de programação (COSTA, 2007).

Uma aplicação Web é executada em um servidor, que nada mais é do que um software que responde às requisições HTTP (*HyperText Transfer Protocol*) de um navegador. Para desenvolver uma aplicação para Web, deve-se considerar sua infraestrutura, que é diferenciada dos servidores de aplicações locais (COSTA, 2007).

## 2.3 Desenvolvimento para a Plataforma Web Usando a Tecnologia PHP

Uma das principais linguagens de desenvolvimento web é o PHP (*PHP: Hypertext Preprocessor*) (2011). Trata-se de uma linguagem de fácil aprendizado que, por poder ser embutida em código HTML (*HyperText Markup Language*), permite um ágil desenvolvimento de páginas.

Criada em 1995 por Rasmus Lerdorf, tinha como objetivo inicial a geração de *scripts Perl* para gerar estatísticas de acesso para seu currículo *online*. Era, inicialmente, conhecida por *Personal Home Page Tools* mas logo foi disponibilizada uma nova versão, bem como seu código fonte em C, chamada PHP/FI (*Personal Home Page/Forms Interpreter*), que já incluía algumas das funcionalidades básicas do PHP conhecido atualmente.

Assim como o Perl, o PHP/FI possuía a sintaxe já embutida no HTML e a interpretação de variáveis provenientes de formulários era automática, mas era mais limitado e sua sintaxe até um pouco inconsistente (PHP, 2011).

Em novembro de 1997 surgiu o PHP/FI 2.0. Segundo o *site* oficial do PHP (2011), milhares de desenvolvedores contribuíram com pequenos códigos que eram incorporados ao projeto, mas logo as primeiras versões *alphas* do PHP 3.0 substituíram de vez o PHP/FI 2.0.

```

1 <!--include /text/cabecalho.html-->
2
3 <!--getenv HTTP_USER_AGENT-->
4 <!--ifsubstr $exec_result Mozilla-->
5 Olá, você está usando Mozilla!<p>
6 <!--endif-->
7
8 <!--sql database select * from tabela where usuario='$usuario'-->
9 <!--ifless $numentradas 1-->
10 Desculpe, o registro não existe<p>
11 <!--endif exit-->
12 Seja bem vindo <!--$user-->!<p>
13 Você tem <!--$index:0--> creditos restantes em sua conta.<p>
14
15 <!--include /text/rodape.html-->

```

Figura 1 - Exemplo de código PHP/FI (PHP, 2011).

Andi Gutmans e Zeev Surasku desenvolveram, em 1997, o PHP 3.0, que surgiu da descoberta de que o PHP/FI 2.0 poderia ajudá-los a construir suas próprias aplicações de *e-commerce* para um projeto da universidade, e é a partir dessa versão que o PHP foi se aproximando do formato como é conhecido atualmente. Andi Gutmans, juntamente com Rasmus Lerdorf e Zeev Surasku anunciaram o PHP 3.0 como a nova versão oficial do PHP/FI 2.0 e descontinuaram este último.

A versão 3.0 do PHP possui como principais características o suporte à orientação objeto, uma forte capacidade de extensibilidade e uma sintaxe muito mais poderosa. Este conjunto de características contribuiu muito para que o PHP se tornasse uma linguagem de grande sucesso. Outro fator que certamente impactou no sucesso dessa linguagem foi a alteração do nome, significando *PHP: Hypertext Preprocessor*, e assim, removendo a impressão de uso limitadamente pessoal. A versão 3.0 foi lançada em junho de 1998.

Apesar de todo o sucesso do PHP 3.0, que foi projetado para melhorar a performance de aplicações complexas e melhorar a modularidade do código base do PHP, a versão 3.0 não foi projetada para trabalhar eficientemente com aplicações muito complexas.

Uma nova *engine* foi introduzida em meados de 1999, a *Zend Engine*, que obteve sucesso em trabalhar com aplicações complexas. O PHP 4.0, baseado nesta *engine* e acompanhado por uma série de novas características, foi oficialmente lançado em maio de 2000, quase dois anos após sua versão anterior. Dentre as

características incluídas nesta versão, destaca-se o suporte para muitos servidores Web, sessões HTTP, buffer de saída, com maneiras mais seguras de manipular entradas de usuários e muitas construções novas na linguagem.

De acordo com o manual do PHP (2011), atualmente o número de servidores que possuem o PHP instalado, representa vinte por cento dos domínios da internet. A equipe de desenvolvimento do PHP possui dezenas de desenvolvedores, bem como dezenas de outros que trabalham com projetos relacionados ao PHP, como, por exemplo, a documentação do projeto e o PEAR.

O PHP 5 foi lançado em julho de 2004, após vários *pré-releases* e um longo desenvolvimento. A principal característica desta versão é a introdução do *core*, a *Zend Engine 2.0* com um novo modelo de orientação a objetos, além de várias outras características.

### 2.3.1 Utilizando o PHP

Em todo início de projeto, uma das principais decisões a ser tomada é a escolha da tecnologia a ser empregada. Muitos fatores influenciam essa escolha, desde o conhecimento dos desenvolvedores até questões como o custo-benefício de se optar por uma determinada tecnologia em detrimento de outra.

A escolha da tecnologia que será utilizada no desenvolvimento de um software é muito importante. Essa escolha deve ser feita levando em consideração o tipo de aplicação que será criada e os recursos que a mesma oferece para o seu desenvolvimento e manutenção.

Como é possível observar na Figura 2, o funcionamento do PHP depende de um servidor interpretar o código PHP, transformando-o em código HTML para ser exibido no navegador.

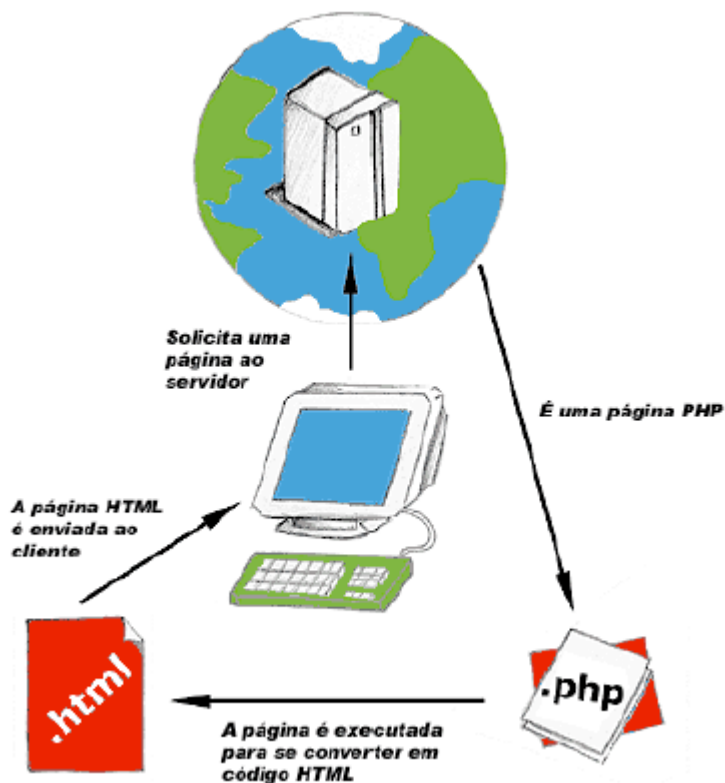


Figura 2 - Funcionamento do PHP <sup>4</sup>

### 2.3.2 Validação *Client-Side* com *JavaScript*

*JavaScript* é a principal linguagem *client-side* utilizada na web atualmente. Diz-se que uma linguagem é *client-side* quando é executada diretamente na máquina do usuário, sem a necessidade de enviar uma requisição ao servidor e aguardar uma resposta (POWELL; SCHNEIDER, 2004).

Trata-se de uma linguagem que abrange uma enorme gama de aplicações, desde simples validações de formulários até interfaces complexas de usuário. Mas como muitas de suas funcionalidades ainda nem foram descobertas pelos usuários, de modo geral sua capacidade é menosprezada.

O *JavaScript* pode ser usado para manipular todas as marcações de documentos em que está inserido e, com isso, está se tornando uma tecnologia Web *client-side* de primeira classe, atuando ao lado de (X)HTML, CSS e XML, à medida em que os desenvolvedores vão descobrindo seu real poder.

<sup>4</sup> Fonte: [http://feliperibeiro.com/downloads/slides\\_php\\_encomp\\_felipe\\_ribeiro.pdf](http://feliperibeiro.com/downloads/slides_php_encomp_felipe_ribeiro.pdf)

Existem quatro maneiras de se incluir o *JavaScript* em um documento, sendo elas: escrever o *JavaScript* no próprio corpo do documento, utilizando o elemento `<script>`; escrever em um arquivo externo e utilizar o atributo `src` do elemento `<script>` para ligá-lo ao documento; utilizar um manipulador de eventos como o `onclick`; e, por fim, via a pseudo-URL `javascript:` referenciada por um *link*.

Neste momento, é importante discorrer um pouco a respeito do element `<script>`, o primeiro método para incluir o *JavaScript* no HTML ou XHTML.

O elemento `<script>` é o responsável por informar o navegador que todo o texto contido entre seu início e fim deve ser interpretado como uma linguagem de *script*. Como padrão dos navegadores, o conteúdo do elemento *script* é *JavaScript*, porém, é possível que o navegador suporte outras linguagens como o VBScript que é suportado pela família de navegadores do Internet Explorer. O atributo *language*, do elemento `<script>`, é o responsável por informar ao navegador qual linguagem deve ser interpretada no trecho em questão, podendo-se ver um exemplo dessa sintaxe no modelo abaixo:

```
<script language="JavaScript">  
</script>
```

No exemplo mostrado, a linguagem a ser interpretada é o próprio *JavaScript*, mas outras linguagens também são aceitas, como, por exemplo, VBScript que seria indicado no atributo *language* como VBS. O navegador simplesmente ignora o conteúdo do elemento Script quando não entende o valor do atributo *language*.

Outro atributo do elemento script é o *type*. Na prática, ele não é tão comum quanto o atributo *language*. Para aproveitar a utilidade do atributo *language* enquanto se respeita os padrões do elemento `<script>`, deve-se considerar o uso de ambos:

```
<script language="JavaScript" type="text/javascript">  
</script>
```

Em alguns casos isso não funciona, pois o navegador tende a respeitar o atributo *type* em detrimento do *language*. Além disso, a página não será validada de acordo com os padrões do XHTML, uma vez que o atributo *language* não faz parte

do padrão definido pela W3C<sup>5</sup>. Dessa forma, usar o atributo *type* é a melhor opção, a não ser que se tenha uma razão específica para utilizar o atributo *language*.

O elemento `<script>` pode ser utilizado da maneira que se desejar. Os documentos serão lidos e, possivelmente executados, na ordem em que forem encontrados, a menos que a execução do script seja adiada.

Abaixo, pode-se ver um exemplo de três scripts sendo executados em sequência:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
  <title>JavaScript e a tag script</title>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body>
  <h1>Pronto para iniciar</h1>
  <script type="text/javascript">
    alert("Executa o primeiro script");
  </script>
  <h2>Executando...</h2>
  <script type="text/javascript">
    alert("Executa o segundo script");
  </script>
  <h2>Keep running</h2>
  <script type="text/javascript">
    alert("Executa o terceiro script");
  </script>
  <h1>Para!</h1>
</body>
</html>
```

Ao executar o *script* acima em diversos navegadores, nota-se que a ordem de execução não é sempre a mesma, ou seja, cada navegador tem sua maneira de lidar com o *JavaScript*.

Ainda sobre a posição em que será inserido o elemento `<script>`, um bom local é dentro da *tag* `<head>` do (X)HTML, pois, devido à sequência natural dos documentos web, o conteúdo da *tag* `<head>` é lido no início, assim, os *scripts* ali contidos são, geralmente, referenciados no corpo do documento. Normalmente, scripts inseridos na *tag head* são usados para definir variáveis ou funções que podem ser usadas mais tarde no documento. Segue um exemplo de como um *script* define uma função que depois é chamada, no corpo do documento, pelo *script* contido no bloco `<script>`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
```

---

<sup>5</sup> <http://validator.w3.org/>

```

<title>JavaScript in the Head</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
<script type="text/javascript">
    function alertTest()
    {
        alert("Perigo! Perigo! JavaScript a frente!");
    }
</script>
</head>
<body>
    <h2 align="center">Script in the Head</h2>
    <hr />
    <script type="text/javascript">
        alertTest();
    </script>
</body>
</html>

```

Uma das grandes vantagens de se usar o *JavaScript* é a possibilidade de criar páginas mais interativas. Para isso, pode-se utilizar comandos que esperam o usuário realizar uma determinada ação. Para especificar tais *scripts*, existem vários manipuladores de eventos, geralmente definindo um atributo do (X)HTML que referencie o *script* que, por sua vez, será executado em resposta a um evento do usuário, por exemplo, *onclick*, *ondblclick* e *onmouseover*.

Abaixo, um exemplo simples de como um botão pode reagir a um clique:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
<title>JavaScript and HTML Events Example</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
</head>
<body>
    <form action="#" method="get">
        <input type="button" value="Enviar" onclick="alert('Olá JS!');" />
    </form>
</body>
</html>

```

Praticamente todos os elementos do HTML podem receber um manipulador de eventos, geralmente todas as *tags* que possuem representação visual.

Como já foi dito, outra forma de incluir um *script* em um documento HTML é ligar um arquivo de extensão js utilizando o atributo *src* do elemento `<script>`. Como no exemplo abaixo:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
<title>Event Trigger Example using Linked Script</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
<script type="text/javascript" src="perigo.js"></script>
</head>
<body>
    <div align="center">

```

```

    <form action="#" method="get">
      <input type="button" value="Não clique!" onclick="alertTest();" />
    </form>
  </div>
</body>
</html>

```

Note que o atributo `src` aponta para o arquivo “perigo.js”. Este valor é o caminho para o *script* externo. Neste caso, o arquivo está no mesmo diretório do documento, mas poderia estar em uma URL absoluta como, por exemplo, `http://www.exemplojavascript.com.br/scripts/perigo.js`. Independentemente da localização do arquivo, o conteúdo do mesmo deve ser o código *JavaScript* para executar a ação solicitada pelo manipulador de eventos, como no exemplo seguinte:

```

function alertTest()
{
    alert("Perigo! Perigo!");
}

```

O benefício de se usar arquivos externos de *scripts* é que eles separam a lógica da estrutura e da apresentação da página. Com um *script* externo, pode-se facilmente referenciá-lo a partir de inúmeras páginas. Isso faz com que a manutenção do código seja mais fácil, pois, quando necessário, basta alterar em um determinado lugar para que a alteração tenha efeito em várias páginas. Além disso, navegadores podem armazenar *scripts* externos em *cache* para aumentar a velocidade de acesso a uma página evitando *downloads* extras a cada vez que tiver que acessar o mesmo *script*.

Outra forma ainda de utilizar *JavaScript* em um documento é chamá-lo via uma pseudo-URL. As pseudo-URLs têm a capacidade de atuar diretamente como *JavaScript* em qualquer lugar do documento, sem a necessidade do elemento `<script>`. Abaixo, um exemplo de como utilizar uma pseudo-URL:

```

<a href="javascript: alert('Sou um script de pseudo-URL');">Clique aqui</a>

```

O *JavaScript* vem sendo amplamente utilizado em aplicações *web*. Muitos acreditam que sua utilização não passa de validações de formulários e ações pontuais disparadas com algum evento realizado pelo usuário. Mas, como afirma Powell e Schneider (2004), o *JavaScript* não se limita a isso, seu uso é bastante útil desde em pequenas ações como as citadas, até na criação de complexas interfaces de usuário.

Por todas as suas características, o *JavaScript* mostrou-se uma excelente opção para realizar as validações necessárias na página do inventário antes mesmo de enviar os dados para serem salvos.

O desenvolvimento de sistemas vive em constante mudança. Não é diferente com o desenvolvimento de sistemas para web. Como a linguagem PHP é uma linguagem *server-side*, ou seja, interpretada em um servidor, é necessária a troca de comunicação entre o computador do usuário e o servidor para que a página seja gerada e apresentada corretamente.

Há poucos anos, a cultura do usuário ainda não “exigia” que a aplicação web fosse tão dinâmica a ponto de atualizar seus dados sem nem mesmo piscar a página. Era permitida, então, a utilização de PHP puro para construção de aplicações, onde a cada requisição, cálculos ou acesso ao banco de dados, uma requisição era enviada ao servidor, processada e só então retornada ao usuário. Dessa forma, o tempo entre enviar uma requisição e receber uma resposta era completamente aceitável.

A própria evolução das tecnologias utilizadas foi criando uma nova cultura nos usuários da web, que já não têm mais a antiga paciência de esperar. Querem tudo no mesmo momento em que solicitaram.

### 2.3.3 Utilizando *trigger* para atualizar o estoque

Existem diversas formas de se controlar o estoque de produtos em um sistema. No sistema iBIsaúde a quantidade de cada produto era calculada com base nos movimentos de entradas e saídas. Para a geração de um relatório de estoque de todos os produtos, era necessário percorrer todos os movimentos de entradas e saídas de cada produto, para realizar o cálculo da quantidade disponível.

Para desenvolver o novo sistema de inventário do iBIsaúde, uma das decisões tomadas consistiu na determinação de que o estoque não poderia mais ser calculado dessa forma, visto que esse cálculo era muito demorado, o que comprometia seriamente a performance do sistema.

A partir dessa decisão, iniciou-se um estudo para decidir qual a melhor maneira de modificar tal cenário, identificar qual ofereceria menor impacto e quais seriam as consequências.

Como resultado desse estudo, e sabendo que *triggers* são procedimentos armazenados que são acionados em determinado momento, vinculados a algum evento, tais como, inserções, atualizações ou exclusões (ABRANCHES, 2008), foi decidido utilizar um *trigger* para atualizar a quantidade em estoque a cada movimentação gerada no sistema, seja de entrada ou saída. Dessa forma, tanto nos relatórios quanto no módulo de inventário, bastaria selecionar a quantidade de cada produto diretamente na tabela de estoque.

Para desenvolver a função que seria disparada pelo *trigger*, de acordo com a documentação do PostgreSQL (2009), poderia ter sido utilizada a linguagem C ou uma das várias linguagens procedurais disponíveis. Foi utilizada, então, a linguagem PL/pgSQL e sua escolha se deu, por ser uma linguagem suportada nativamente pelo SGBD utilizado, o PostgreSQL. Cabe ressaltar que, caso a linguagem escolhida não fosse suportada pelo SGBD, a mesma poderia ser instalada no SGBD e, assim, estaria disponível para o desenvolvimento de funções.

O *trigger* pode ser definido, utilizando o comando *CREATE TRIGGER*, para executar a função antes e depois de uma operação, podendo ser disparado a cada linha modificada ou a cada instrução SQL. Existe um vínculo direto entre o *trigger* e a função que o mesmo dispara, mas isso não impede que uma função de *trigger* seja ativada por vários *triggers* (PostgreSQL, 2009).

O que define se o *trigger* será disparado a cada linha ou a cada instrução SQL é o seu tipo. Um *trigger* do tipo linha executa a função para cada linha modificada, portanto, se uma instrução SQL não afetar nenhuma linha, a função do *trigger* não será executada. Por outro lado, um *trigger* do tipo instrução é disparado sempre que a instrução SQL é executada, mesmo que esta não afete nenhuma linha.

Caso uma determinada instrução dispare mais de um *trigger*, por exemplo, um por instrução e outro por linha, a ordem de execução de suas funções será esta: o *trigger* por instrução BEFORE tem precedência sobre um *trigger* por linha BEFORE, que por sua vez, tem precedência sobre *trigger* por linha AFTER, o qual tem precedência sobre um *trigger* por instrução AFTER (PostgreSQL, 2009).

Alguns cuidados referentes ao retorno de uma função de *trigger* devem ser tomados como, por exemplo, uma função de *trigger* por instrução deve sempre retornar NULL. Isso faz com que a operação salte para a linha corrente, instruindo ao executor a não executar a operação no nível de linha que chamou o *trigger*.

Quando se trata de *trigger* por linha de INSERT ou UPDATE, a linha retornada se torna a linha que será inserida ou que substituirá a linha que está sendo modificada. Caso a intenção não seja causar esse comportamento, o *trigger* deve retornar a mesma linha que recebeu (ou seja, a linha NEW para INSERTs e UPDATEs e a linha OLD para DELETEs). Os *triggers* por linha, executados depois da modificação, ignoram o retorno, dessa forma, não importa qual o retorno da função (PostgreSQL, 2009).

Quando há mais de um *trigger* para o mesmo evento na mesma relação, a ordem de execução dos mesmos é a alfabética. No caso de *trigger* BEFORE, o retorno de uma função é utilizado como entrada para a próxima. Se algum dos *triggers* BEFORE retornar NULL a operação é abandonada e os *triggers* seguintes não são disparados (PostgreSQL, 2009).

A utilização de *triggers* BEFORE ou AFTER depende do resultado desejado. Tipicamente, os *triggers* BEFORE são utilizados para verificar ou modificar os dados da linha que será inserida ou atualizada, enquanto os que disparam AFTER são mais utilizados para propagar as atualizações para outras tabelas ou, ainda, verificar a consistência dos dados de outras tabelas. Caso não exista nenhum motivo específico para se optar pelo *trigger* BEFORE ou AFTER, o *trigger* BEFORE é o mais indicado, pois a informação não precisa ser salva até o fim da instrução.

Um problema possível de acontecer quando se executa uma função de *trigger* é o conhecido por cascadeamento de *triggers*. Acontece quando a função do *trigger* realiza uma operação que dispara outro *trigger*. É responsabilidade do programador do *trigger* evitar que o disparo de *triggers* execute recursões infinitas, como no caso de um *trigger* de INSERT que executa uma inserção na mesma tabela, direta ou indiretamente.

A regra básica de visibilidade dos dados por uma função de *trigger* é que as mudanças realizadas pela instrução SQL que o acionou são enxergadas apenas nos casos em que o momento de execução definido no *trigger* é depois (AFTER), embora as operações SQL executadas nas funções de *trigger* por linha BEFORE podem enxergar os efeitos de mudanças realizadas no registro anterior. Como a ordem dessas mudanças, em geral não é previsível, deve-se ter muita cautela nessa situação (MATTOSO *et al.*, 2005).

A sintaxe de criação de um *trigger* é apresentada na Figura 3.

```

1 CREATE TRIGGER nome { BEFORE | AFTER } { evento [OR ...] }
2     ON tabela [ FOR [ EACH ] { ROW | STATEMENT } ]
3     EXECUTE PROCEDURE nome_da_funcao()

```

Figura 3 - Sintaxe da criação de um trigger genérico (PostgreSQL, 2009).

Os argumentos esperados em um comando de criação de trigger são: nome, que é a identificação do *trigger*; *before* ou *after* indica se a função será executada antes ou depois do evento; evento determina a situação em que a função será disparada (INSERT, UPDATE ou DELETE); tabela faz a associação do *trigger* com uma tabela; *row* ou *statement* define se a função será disparada a cada linha ou a cada instrução SQL, e, no caso de não ser informado, o valor padrão, FOR EACH STATEMENT, é assumido; e nome\_da\_funcao especifica qual função deve ser executada no momento de disparo do *trigger*.

Um exemplo de criação de *trigger* pode ser visto na Figura 4.

```

1 CREATE TRIGGER tg_insert_tabela_b AFTER INSERT
2     ON tabela_a FOR EACH ROW
3     EXECUTE PROCEDURE fn_insert_tabela_b();

```

Figura 4 - Exemplo de criação de trigger (PostgreSQL, 2009).

A função executada por um *trigger* é comumente chamada de função de *trigger*. Trata-se de um tipo especial de função, pois existe uma série de particularidades que as diferenciam das funções comuns: o tipo do retorno deve ser *trigger*; a linguagem em que ele é escrito deve ser especificado após a palavra LANGUAGE e seu corpo apresentado entre aspas ou cifrão depois da palavra AS, conforme exemplo da Figura 5.

```

1 CREATE OR REPLACE FUNCTION fn_insert_tabela_b()
2 RETURNS trigger LANGUAGE plpgsql
3 AS
4 $body$
5     begin
6         insert into tabela_b (valor, numero)
7         values (new.valor, new.codigo);
8         return new;
9     end;
10 $body$;

```

Figura 5 - Exemplo de função de trigger (PostgreSQL, 2009).

No exemplo mostrado pela Figura 4, pode-se notar a criação de um *trigger* que dispara a função da Figura 5 e, por sua vez, executa uma inserção na tabela 'tabela\_b' a cada linha inserida na tabela 'tabela\_a', utilizando os valores do comando de inserção da tabela 'tabela\_a'.

### 3 DESENVOLVIMENTO DO MÓDULO DE INVENTÁRIO IBISAÚDE

Este capítulo apresenta uma visão geral do módulo de inventário básico do sistema iBIsaúde; a visão geral do módulo de inventário desenvolvido com controle de lote e validade; e as principais características do módulo de inventário.

#### 3.1 Visão Geral do Módulo de Inventário básico

O módulo de inventário do sistema iBIsaúde sempre foi um módulo muito importante para o funcionamento de todo o sistema, pois é por meio do inventário que o estoque é atualizado para garantir que a quantidade de produtos que consta no sistema está de acordo com a quantidade de fato existente em estoque.

No módulo básico de inventário, para se obter o valor do estoque, eram contados todos os itens de um determinado produto e o valor correspondente era cadastrado no sistema. Caso a quantidade informada ao sistema não fosse igual à que o sistema calculava, eram geradas movimentações para que os ajustes necessários fossem realizados.

A Figura 6 mostra a antiga tela do módulo de inventário:

| Produto                       | Quantidade           |
|-------------------------------|----------------------|
| TERBUTALINA 0,5MG/ML - AMPOLA | <input type="text"/> |
| TETRACICLINA 500MG - CAPSULA  | <input type="text"/> |
| TIAMINA 300 MG COMPRIMIDO     | <input type="text"/> |
| TIMOLOL 0,5% COLIRIO          | <input type="text"/> |
| VITAMINA C 100MG/ML - AMPOLA  | <input type="text"/> |






Figura 6 - Módulo básico de Inventário

Este modelo funcionou por bastante tempo até surgir a necessidade de realizar um controle minucioso dos produtos para evitar perdas por validade vencida, ou ter a falsa idéia da quantidade disponível de determinado produto. Por exemplo, o sistema podia apontar que existiam quinhentas unidades do produto Y, descartando a necessidade de compra, mas quando verificado no estoque, percebia-se que

todas as unidades do produto estavam vencidas, constatando-se, assim, a falta do mesmo.

Para controlar o prazo de validade dos produtos em estoque, além de sua quantidade, foi solicitado o controle do produto por lote e data de validade, visto que todos os produtos de um determinado lote apresentam o mesmo prazo de vencimento.

Realizando-se o controle dos produtos por lote e prazo de validade no sistema, o módulo de inventário também deveria permitir tal controle, visto que isso é parte de fundamental importância para o correto controle do estoque.

### 3.2 Visão Geral do Módulo de Inventário com controle de Lote e Validade

O controle por lote e validade surge como a solução para os problemas apontados na seção anterior. Para desenvolver o módulo de inventário, tão importante para o controle do estoque, o primeiro passo foi identificar as imperfeições existentes no módulo em operação, para providenciar as devidas correções no módulo que seria desenvolvido.

O principal recurso introduzido foi o acréscimo dos campos para registro de lote e validade no momento de cadastrar a contagem de cada produto. As outras alterações, embora não sejam visíveis ao usuário do sistema, são ainda mais importantes para o funcionamento do novo módulo. É o caso da criação de novos campos e de funções de atualização das tabelas do banco de dados.

A Figura 7 apresenta a nova interface do módulo inventário. Agora é possível introduzir várias linhas para cada produto, sendo que cada uma representa as informações de um determinado lote do produto.

| Produto  | Quantidade | Lote     | Validade   |   |
|--|------------|----------|------------|---|
| AAS 100 MG - COMPRIMIDO                                      | 83720      | 0656097  | 30/12/2012 | + |
|  | 302000     | 0656098  | 30/12/2012 |   |
|  | 14280      | 0656099  | 30/12/2012 |   |
|  | 176550     | 100817   | 30/09/2012 |   |
|  | 19900      | 101040   | 30/10/2012 |   |
|  |            |          |            |   |
| ACETATO MEDROXIPROGESTERONA 150MG/ML - AMPOLA (DEPO-PROVERA) | 377        | 2933491  | 30/09/2012 | + |
|  | 130        | 3095981  | 30/09/2012 |   |
|  | 10         | 3342062  | 30/11/2012 |   |
|  | 248        | 281516.1 | 30/03/2012 |   |
|  |            |          |            |   |
| ACIDO FOLICO 5MG - COMPRIMIDO                                | 6000       | 10080777 | 30/08/2012 | + |
|  | 43800      | 102546   | 30/04/2012 |   |
|  | 11420      | 091397   | 30/03/2011 |   |

Figura 7 - Interface do novo módulo de inventário

### 3.3 Principais Características do Módulo de Inventário

Nesta seção são apresentadas as principais características do módulo inventário do sistema iBlsaúde.

#### 3.3.1 Possibilidade de cadastrar diversos lotes e validades de cada produto

Como o requisito principal do novo módulo de inventário do sistema iBlsaúde era controlar o estoque de acordo com o lote e o prazo de validade dos produtos, foram adicionados ao sistema os respectivos campos, como visto na Figura 8 e Figura 7.

| Produto                              | Quantidade  | Lote  | Validade  |
|--------------------------------------|---|---|---|
| AMOXICILINA 500MG - CAPSULA          | <input type="text" value="1500"/><br><input type="text" value="500"/><br><input type="text" value="198000"/><br><input type="text" value="102620"/><br><input type="text"/> | <input type="text" value="107620"/><br><input type="text"/><br><input type="text"/> | <input type="text" value="30/04/2013"/><br><input type="text"/><br><input type="text"/> |
| AMPICILINA 500MG - CAPSULA           | <input type="text"/>  | <input type="text"/>  | <input type="text"/>  |
| ANLODIPINO BESILATO 5MG - COMPRIMIDO | <input type="text"/>  | <input type="text"/>  | <input type="text"/>  |

Botão para excluir a linha inserida.

Botão para incluir uma nova linha.

Figura 8 - Funcionamento do módulo de inventário

Nota-se, na Figura 8, que alguns produtos já são carregados com os campos visíveis e preenchidos. Esses são os lotes e validades que apresentam quantidade maior que zero. Se houver necessidade de cadastrar um novo lote do produto, basta pressionar o botão verde com o símbolo “+”. No caso de inserção de mais campos do que o necessário, basta removê-los pressionando o botão vermelho com o símbolo “-”.

Sabendo que o *JavaScript* é uma boa opção para realizar validações *client-side*, foi utilizado para impedir a digitação de caracteres proibidos no campo quantidade, onde só é permitido caracteres numéricos; foi utilizado também para realizar a validação da data digitada no campo validade, onde a mesma não pode ser inferior à data atual e ainda deve ser uma data válida; outra utilização do *JavaScript* no desenvolvimento do módulo de inventário foi para realizar a inserção dos campos dinamicamente.

O processo de realização do inventário periódico consiste em: escolher o grupo de produtos a serem contados; selecionar o setor em que ocorrerá a contagem; informar o responsável pelo inventário; informar qual equipe realizará a contagem; realizar a contagem manual; digitar no sistema os valores obtidos; salvar os dados digitados; e, por fim, gerar a movimentação.

No momento de cadastrar um produto no sistema é definido um grupo para o mesmo. Os grupos diferenciam os produtos, o que facilita a realização do inventário periódico, permitindo que o mesmo seja realizado apenas sobre uma parte dos produtos do estoque, não sendo necessária, assim, a contagem de todo o estoque.

O setor informado no cadastro de um inventário é o local onde será realizada a contagem do estoque. Para que um setor possa ser selecionado na criação de um inventário, o mesmo deve estar marcado como Centro Estocador<sup>6</sup>.

Os campos destinados ao responsável e equipe do inventário, conforme mostra a Figura 9, são apenas informativos, servindo de controle para o próprio responsável.

A imagem mostra uma interface de usuário para o 'Cadastro de Inventário'. O formulário contém os seguintes campos e botões:

- Data:** Campo de texto com o valor '27/02/2011'.
- Grupo de Produto:** Menu suspenso com o valor 'IMPRESSOS'.
- Setor:** Menu suspenso com o valor 'ALMOXARIFADO'.
- Responsável:** Campo de texto vazio.
- Equipe:** Campo de texto vazio.
- Botões:** 'ADICIONAR' (com ícone de adicionar) e 'VOLTAR' (com ícone de voltar).

**Figura 9 - Criação do inventário**

Após a contagem manual dos produtos, é tarefa do responsável pelo inventário digitar no sistema os dados obtidos e salvá-los. O sistema irá salvar todos os dados digitados em uma tabela do inventário. Nesse momento, o estoque ainda não foi alterado.

A partir do momento em que o responsável pelo inventário seleciona a opção “Gerar Movimentação”, o sistema verifica o estoque de cada produto e faz o ajuste do mesmo. Para fazer esse ajuste, o sistema verifica a diferença entre a quantidade contida na tabela do inventário e a quantidade contida na tabela de estoque e gera uma movimentação de entrada ou de saída, de acordo com esse valor.

---

<sup>6</sup> Um setor é chamado de centro estocador se possui um estoque.

### 3.3.2 Diferenciação entre produtos com as informações de lote e validade e produtos sem tais informações

Com o desenvolvimento do novo módulo de inventário, foi necessário diferenciar os produtos quanto à necessidade de controle ou não de lote e prazo de validade. Foi decidido realizar esse controle diretamente na tabela de produtos e, para tal, um novo campo foi adicionado à mesma.

Com esse campo preenchido para todos os produtos, o módulo de inventário apresenta os campos de lote e validade apenas para os produtos marcados como sendo necessário, indiferente ao grupo ao qual o produto pertence. A Figura 10 apresenta o exemplo de um inventário em que os produtos não necessitam do controle por lote e prazo de validade.

| Produto  | Quantidade           | Lote | Validade             |
|--|----------------------|------|----------------------|
| BL ACIDO URICO ELEVADO 100 X 1 - UNID (NASF)       | <input type="text"/> |      | <input type="text"/> |
| BL ACOMPANHAMENTO HIPERTENSO 100 X 1 - UNID        | 29                   |      |                      |
| BL ACOMPANHAMENTO DIABETICO 100 X 1 - UNID         | 60                   |      |                      |
| BL ACOMPANHAMENTO GESTANTE 100 X 1 - UNID (PSF)    |                      |      |                      |
| BL ACOMPANHAMENTO PESSOAS HANSEIASE 100 X 1 - UNID | <input type="text"/> |      |                      |
| BL ACOMPANHAMENTO TUBERCULOSE 100 X 1 - UNID       |                      |      |                      |

**Figura 10 - Exemplo de inventário de produtos que não necessitam do controle por lote e validade**

Outra diferença entre um produto que necessita do controle por lote e prazo de validade consiste no fato de que, ao ser acionado o botão adicionar (com o símbolo "+"), o mesmo é imediatamente desabilitado, pois não poderia ser gerado mais de um registro por produto.

### 3.3.3 Geração da movimentação (Entrada ou Saída) para atualizar o estoque

Como citado na seção 3.3.1, a geração da movimentação atualiza o estoque dos produtos contidos na tabela do inventário.

Essa atualização acontece porque existe um *trigger* do tipo por linha (AFTER) que dispara uma função depois de cada inserção na tabela do movimento, e é essa função a responsável por efetuar as atualizações na tabela de estoque. Outros dois

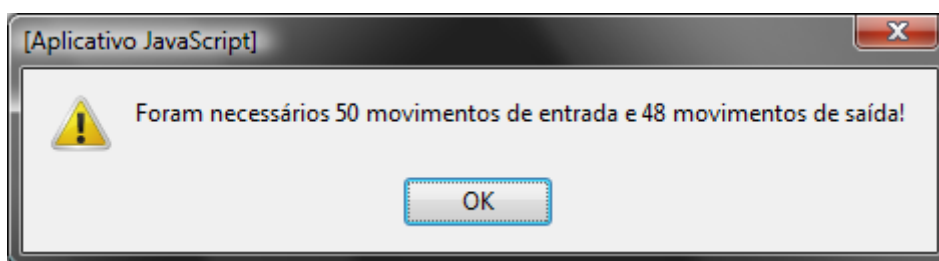
*triggers* foram necessários para atualizar o estoque no caso de exclusões e alterações na tabela do movimento.

O botão “Gerar Movimentação” pode ser visto na Figura 11 e retorna ao usuário uma mensagem informando quantos movimentos de entrada e saída foram gerados como mostrado na Figura 12.

| Resultado |                    |             |                                  |              |              |
|-----------|--------------------|-------------|----------------------------------|--------------|--------------|
| Código    | Data do Inventário | Responsável | Equipe                           | Grupo        | Setor        |
| 114       | 21/02/2011         | VIVIANI     | IVANDER, TIAGO, LOURDES E RENATO | MEDICAMENTOS | ALMOXARIFADO |

ADICIONAR GERAR MOVIMENTAÇÃO

**Figura 11 - Tela de Geração da Movimentação**



**Figura 12 - Mensagem retornada ao usuário no botão Gerar Movimentação**

Após a geração da movimentação, não é permitida nenhuma outra alteração no inventário.

#### 4 CONCLUSÕES E TRABALHOS FUTUROS

Com o desenvolvimento deste trabalho, pode-se concluir que realizar um controle de estoque corretamente é tarefa de fundamental importância para o funcionamento correto do centro estocador, bem como de todo o sistema, gerando economia de tempo, de esforço e, conseqüentemente, redução dos custos para o usuário final, além de propiciar maior confiabilidade e eficácia em tal controle.

Durante a implantação do novo módulo de inventário do sistema iBIsaúde, o cliente se mostrou satisfeito com as alterações realizadas. Três pontos importantes para tal afirmação foram: facilidade no momento de fazer compra de produtos, pois com o controle do lote e prazo de validade dos produtos, ficou mais fácil decidir a quantidade a ser comprada; além disso, tal controle ajuda a evitar a perda de produtos, por vencimento do prazo de validade, organizando a distribuição dos mesmos; e, por fim, o processo de geração de alguns relatórios ficou bem mais ágil devido à implantação do *trigger*.

Durante o desenvolvimento do módulo de inventário do sistema iBIsaúde, algumas experiências destacaram-se: a utilização de funções de *trigger* para alterar a forma de se controlar o estoque; a utilização do *JavaScript* para criar os campos dinamicamente e validar os dados antes de salvá-los; e o aprendizado da regra de negócio de um sistema de controle de estoque.

Como sugestão de trabalhos futuros, existe a possibilidade de desenvolver o inventário utilizando um leitor de código de barras, para eliminar o processo manual da contagem.

Outra possibilidade de trabalho futuro é o desenvolvimento de um método que seja capaz de realizar o inventário de produtos específicos em vez de obrigar o usuário a efetuar a contagem de todos os produtos de um determinado grupo, como acontece atualmente.

## REFERÊNCIAS BIBLIOGRÁFICAS

ABRANCHES, Danilo. **Triggers no PostgreSQL**. Disponível em: <[http://imasters.com.br/artigo/10644/triggers\\_no\\_postgresql](http://imasters.com.br/artigo/10644/triggers_no_postgresql)>. 2008. Acesso em 02 fev. 2011.

COSTA, Carlos J. **Desenvolvimento para Web**. Lisboa: Lusocrédito,Lda, 2007. Disponível em: < [http://books.google.com/books?hl=en&lr=&id=Jn6dTDF-wcsC&oi=fnd&pg=PT3&dq=desenvolvimento+para+web&ots=wJhMOQZ7Xj&sig=C8oNVOnbB9bCVzzwrMbol6hQF\\_w#v=onepage&q&f=false](http://books.google.com/books?hl=en&lr=&id=Jn6dTDF-wcsC&oi=fnd&pg=PT3&dq=desenvolvimento+para+web&ots=wJhMOQZ7Xj&sig=C8oNVOnbB9bCVzzwrMbol6hQF_w#v=onepage&q&f=false)>. Acesso em: 08 fev. 2011.

FILHO, Américo Garcia Parada. **LIVRO DE INVENTÁRIO E SISTEMAS DE CONTROLE DOS ESTOQUES**. Disponível em: <<http://www.cosif.com.br/mostra.asp?arquivo=contabil04inventario>>. Acesso em 09 fev. 2011.

MATTOSO, Marta; ZIMBRÃO, Geraldo; LIMA, Alexandre A. B.; BAIÃO, Fernanda; BRAGANHOLO, Vanessa de Paula; AVELEDA, Albino A.; MIRANDA, Bernardo Faria; ALMENTERO, Bruno Kinder; NUNES, Marcelo. **Análise Funcional do SGBD PostgreSQL. Projeto COPPETEC PESC 5391, Relatório RT-01**. Centro de Tecnologia. Rio de Janeiro. 2005.

PHP - Site oficial PHP. 2011. Disponível em: <[http://www.php.net/manual/pt\\_BR/preface.php](http://www.php.net/manual/pt_BR/preface.php)>. Acesso em 09 fev. 2011.

POWELL,Thomas A.; SCHNEIDER,Fritz. **JavaScript: The Complete Reference**, 2nd edition: The Complete Reference. Complete coverage of the W3C DOM2 standard (Osborne Complete Reference Series). 2004.

PostgreSQL - Documentação do PostgreSQL. 2009. **Tradução da documentação do PostgreSQL para o Português do Brasil**. Disponível em: <<http://pgdocptbr.sourceforge.net/pg80/triggers.html>>. Acesso em 02 fev. 2011.