Rotina Javascript

Com as ações do tipo Rotina no Banco de Dados vimos como estender as funcionalidades das telas do Sankhya-Om com StoredProcedure, maximizando as possibilidades de customização.

Uma opção às ações de Banco de dados são as ações do tipo Script, que através da linguagem JavaScript abrem um novo arsenal de programação, além de suportar as mesmas funcionalidades básicas apresentadas anteriormente. Apesar da semelhança do nome, JavaScript não possui relação com Java.

JavaScript é uma linguagem poderosa e bastante difundida em aplicações Web, sendo fácil obter exemplos e tutoriais de como utilizá-la.

É importante destacar que o JavaScript que usaremos será executado no servidor e não no browser, como é mais comumente usado. Essa diferença faz com que alguns recursos do navegador, como a função alert() e o objeto window não estejam disponíveis para uso em nossos scripts, sendo que esses recursos são de fato do navegador e não da linguagem. No entanto, ao lado do servidor, é possível fazer outras coisas que o JavaScript do browser não consegue, como consultas e alterações no Banco de dados.

Ao utilizar o JavaScript, alguns recursos usados no JavaScript de páginas HTML, como aqueles que rodam no ambiente do navegador, não estão disponíveis para serem utilizados em ações. Dessa forma, como na maioria das linguagens, é possível criar variáveis, usar estruturas como laços de repetição, operadores de comparação, operadores aritméticos, etc. Também estão disponíveis a maioria das funções globais, como parseInt, isNaN, etc. Além disso, você também pode declarar suas próprias funções e invocá-las livremente.

> 📘 **Observação**
>
> A maioria das fontes de estudos misturam os conceitos da linguagem JavaScript e do mecanismo que o navegador usa para interpretar o script das paginas HTML. Tente não se confundir, pois o objetivo dos nossos scripts não é tratar HTML.

Portanto, teremos disponível além do JavaScript, a sintaxe básica da linguagem com alguns recursos especiais. Assim, teremos:

Além da sintaxe básica da linguagem estão disponíveis alguns recursos especiais. Veja:

Funções

Função novaLinha: novaLinha([nomeDaTabela]) – Função utilizada para inserir um registro novo. Se for informado o nome da tabela, o registro será criado nessa tabela. Caso contrário será criado na tabela à qual pertence a ação. Essa função, retorna um objeto do tipo **Registro**, que será detalhado adiante.

Observação: Quando essa função é chamada, o registro ainda não existe no banco de dados e não estará disponível em consultas até o fim da ação. Mas se durante a execução for necessário antecipar a inclusão, o método save() do registro pode ser acionado. Assim, teremos:

SQL

```
var financeiro = novaLinha("TGFFIN");
//Aqui temos um registro virtual pronto para receber todos os campos do financeiro
//mas ele é criado vazio e não existe no banco de dados.
...
//Depois de informar todos os campos obrigatórios, podemos efetuar a inclusão
financeiro.save();
```

Função getParam: getParam(nomeDoParametro) – Função responsável por retornar o valor dos parâmetros que você informar. Segue o mesmo principio discutido no exemplo da StoredProcedure, porém, mais simples, pois seu único argumento é o nome do parâmetro, sendo ele:

SQL

```
var codigoDoVeiculo = getParam("CODVEICULO");
```

Função confirmar: confirmar(titulo, texto, indice) – Essa função exibe um diálogo, em que você terá a opção de prosseguir ou cancelar a ação. É muito útil em situações onde o resultado da ação pode trazer consequências imprevistas. Dessa forma, temos o referido diálogo:

SQL

```
confirmar("Valor do título zerado", "Desdobramento igual a zero. Deseja continuar?",
1);
```

Função confirmarSimNao: confirmarSimNao(titulo, texto, indice) – Funcionamento semelhante à confirmar, porém nessa além de cancelar a execução, o usuário pode optar por "**Sim**" ou "**Não**". Ela pode retornar true ou false.

SQL

```
if(confirmarSimNao("Registro inconsistente", "Localizado um registro inconsistente.
Deseja corrigi-lo?', 1)){
   ...
} else {
   ...
}
```

Função email: email(titulo, mensagem, destinatarios) – Essa função adiciona uma mensagem a ser enviada por email, uma vez que a lista de destinatários deve ser separada por vírgula.

Função mostraErro: mostraErro(mensagem) – Esta irá interromper a execução da ação, mostrando uma mensagem de erro.

Função getUsuarioLogado: getUsuarioLogado() – Retorna o código do usuário logado.

Função getQuery(): – Cria um objeto capaz de executar consultas ou mesmo alterar o estado do banco de dados. O objeto retornado é um QueryExecutor. Observe detalhes desse objeto abaixo:

SQL

```
var query = getQuery();

//Os parâmetros DEVEM ser incluidos antes de executar a consulta.
query.setParam("CODVEICULO", getParam("CODVEICULO"));
query.nativeSelect("SELECT * FROM TGFVEI WHERE CODVEICULO = {CODVEICULO}");

while(query.next()){
    var placa = query.getString("PLACA");
    ...
}
```

Função newJava: newJava(classname) – Por meio deste, é possível criar instâncias de objetos java para suprir uma eventual demanda não atendida pela estrutura JavaScript. Por exemplo, veja como formatar uma data:

SQL

```
//Instanciamos um objeto SimpleDateFormat do java
var simpleDateFormat = newJava("java.text.SimpleDateFormat");

//determinamos o padrão de formatação pelo método applyPattern. Mais detalhes podem
ser
//vistos em
http://http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html
simpleDateFormat.applyPattern("dd/MM/yyyy");

//utilizando o método format, convertemos uma data em string, formatando de acordo
com o padrão determinado acima.
var dataFormatada = simpleDateFormat.format(new Date());

//pra finalizar exibimos uma mensagem para ver o resultado da formatação
mensagem = "A data formatada é " + dataFormatada;
```

Função javaClass: javaClass(classname) – Essa função se assemelha à anterior, porém, o retorno é uma Classe java e não uma instância dessa classe. Serve para usar métodos e atributos estáticos das classes Java.

Variaveis de ambiente

- **linhas** (tipo Array): Esse array representa as linhas selecionadas na grade. Os itens desse array são objetos do tipo Registro.
- **linhaPai** (tipo Registro): Linha pai é um objeto do tipo Registro que representa a linha da tela master quando a ação estiver vinculada a uma tela detalhe.
- **mensagem** (tipo String): Se algum valor for atribuído a essa variável, será exibido como uma informação ao final da execução.

Objetos Específicos para ações

- **Registro:** Esse objeto representa a linha de uma tabela. Nele, temos dois métodos, sendo o primeiro deles o "remove()", que não possui retorno e determina que o registro seja excluído. Em seguida, temos o "save()", em que nele, usualmente o registro será salvo de forma automática ao final da ação. Eventualmente, pode-se antecipar esse procedimento manualmente através desse método.

- **setCampo(nomeDoCampo, valorDoCampo)**: Altera o valor de um campo. Esse método não tem retorno.

- **getCampo(nomeDoCampo)**: Retorna o valor de um campo.

- **QueryExecutor:** Objeto responsável por interagir com o Banco de dados. Ele é obtido através da função **getQuery** descrita acima. Veja um exemplo básico de consulta ao banco de dados usando esse objeto:

SQL

```
//Obtemos uma instância do QueryExecutor através da função getQuery
var query = getQuery();

//Se a consulta utiliza parâmetros, devemos informá-los antes de executar
query.setParam("DATAINICIO", "01/09/2013");
query.setParam("DATAFIM", "03/10/2013");

//Executamos a consulta (o nome dos parâmetros deve ser delimitado por {})
query.nativeSelect("SELECT * FROM TGFFIN WHERE DHMOV > {DATAINICIO} AND DHMOV <
{DATAFIM}");

//E criamos um laço para percorrer o resultado, adicionando o número dos títulos em
um array
//para exibir uma mensagem no final.

var titulos = new Array();

while(query.next()){
      //esse método push, adiciona um elemento ao final do array.
      titulos.push(query.getString("NUFIN"));
}

if(titulos.length > 0){
      //o método join do array, concatena os elementos, colocando um separador entre
eles.
```

```
        mensagem = 'Foram localizados os títulos "' + titulos.join(",") + '" no
período pesquisado.';
} else {
        mensagem = "Não existem títulos para o períodos pesquisado";
}
```

Temos a seguir, os métodos do QueryExecutor:

- **nativeSelect(consultaSql)**: Executa a consulta. Sendo que, podem ser usados parâmetros na consulta envolvendo o nome do parâmetro por chaves (“{MEU_PARAMETRO}”). Esse metódo não possui retorno.
- **update(consultaSql)**: Usado para fazer UPDATE ou INSERT no banco. Não há retorno para esse método.
- **setParam(nomeDoParametro, valor)**: Atribui valor aos parâmetros da consulta/alteração. Não possui retorno.
- **next()**: Navega no resultado da consulta. Quando não há mais registros retorna false.
- **getDouble(coluna)**: Retorna o valor da coluna solicitada. Pode ser o índice (1 para a 1ª coluna, 2 para a 2ª, etc.), ou o nome da coluna no banco de dados. O retorno é um número decimal.
- **getInt(coluna)**: Para colunas com conteúdo Inteiro.
- **getString(coluna)**: Para colunas de texto.
- **getDate(coluna)**: Retorna como uma data.
- **getObj()**: Retorna o objeto da tabela (ou com campos selecionados no SQL), já em uma classe, já fazendo os Cast de acordo com o tipo no banco.

Exemplo prático

A seguir, exibiremos um exemplo Prático. Assim, faremos um controle de quilometragem de frota, utilizando uma tela adicional, observe:

Controle de quilometragem (TADCKM) – Tela para lançamento de movimentação de veículos.

SEQUENCIA(PK): Esse campo é a PK da tabela e é auto numerado;

CODVEICULO: Campo importado da **TGFVEI**;

SAIDA: Data e hora de saída do veículo;

CHEGADA: Data e hora de chegada do veículo;

KMINICIAL: Quilometragem do veículo na saída;

KMFINAL: Quilometragem do veículo na chegada;

DISTÂNCIA: Diferença entre a quilometragem de chegada e de saída. Esse campo é do tipo calculado e deve ter a seguinte expressão:

JavaScript

```
if($col_KMINICIAL == null || $col_KMFINAL == null){
     return null;
} else {
     return $col_KMFINAL.subtract($col_KMINICIAL);
}
```

REEMBOLSO: O valor desse campo será calculado com base na distância percorrida multiplicado pelo preço do Km.

Para fazer o cálculo do reembolso, criaremos uma ação do tipo Script com a rotina abaixo:

JavaScript

```
for(var i = 0; i < linhas.length; i++){
    var linha = linhas[i];
    linha.setCampo("REEMBOLSO", linha.getCampo("DISTANCIA") * getParam("PRECOKM"));
}
```

E criamos um parâmetro numérico para representar o preço do KM:

Depois de alguns lançamentos na tela de "Controle de quilometragem", ao executar a ação "Calcular reembolso", nossa ação usa o preço do Km informado por parâmetro e o campo calculado DISTANCIA para determinar o valor do campo REEMBOLSO.

Perceba que utilizamos um campo que não existe no banco de dados, sendo ele o campo calculado DISTANCIA. Isso não seria possível em uma ação do tipo StoredProcedure.

Outra vantagem de ações Script em relação a StoredProcedure, é que não é necessário informar a PK auto-numerada o que não acontece quando inserimos registros diretamente no banco de dados, uma vez que a chave automática é gerada no servidor de aplicação (como foi visto no exemplo da StoredProcedure precisamos fazer um SELECT MAX para determinar a próxima PK disponível). Dessa forma, teremos:

JavaScript

```
var linha = novaLinha();

//O campo SEQUENCIA que é a PK, não precisa ser informado, pois esse campo é auto-
numerado
linha.setCampo("CODVEICULO", 6);
linha.setCampo("SAIDA", "11/09/2012");
linha.setCampo("CHEGADA", "12/09/2012");
linha.setCampo("KMINICIAL", 33250);
linha.setCampo("KMFINAL", 33300);
```

Uma outra ação para essa tela, seria lançar um Título financeiro para o reembolso das despesas com combustível:

Crie uma nova ação do tipo script contendo o seguinte código:

JavaScript

```
//Obtemos uma consulta para buscar os lançamentos
var query = getQuery();

//preparamos a execução da query, incluindo o parâmetro CODVEICULO.
query.setParam("CODVEICULO", getParam("CODVEICULO"));

query.nativeSelect("SELECT * FROM AD_TADCKM WHERE CODVEICULO = {CODVEICULO}");

var vlrDesdob = 0;
while(query.next()){
   var reembolso = query.getDouble("REEMBOLSO");

   //Só permitimos gerar o título quando todos
   //os lançamentos estiverem com reembolso calculado.
   if(reembolso > 0){
       vlrDesdob += reembolso;
   } else {
       mostraErro("O reembolso do lançamento " + query.getInt("SEQUENCIA") + " não
foi calculado ainda.");
   }
}

if(vlrDesdob == 0){
    confirmar("Valor do título zerado", 'O veículo informado não possui lançamentos
para reembolso, o título terá valor de desdobramento igual a zero. Deseja
continuar?', 1);
}

//por questões de desempenho é aconselhavel
//fechar a consulta sempre que ela não for mais necessária.
query.close();

//Solicitamos a inclusão de uma linha no financeiro
var financeiro = novaLinha("TGFFIN");

//Informamos os campos desejados para incluir o financeiro
financeiro.setCampo("VLRDESDOB", vlrDesdob);

financeiro.setCampo("RECDESP", -1);
financeiro.setCampo("CODEMP", 11);
financeiro.setCampo("NUMNOTA", 0);
financeiro.setCampo("DTNEG", "04/10/2012");
financeiro.setCampo("CODPARC", 0);
financeiro.setCampo("CODNAT", 3050200);
financeiro.setCampo("CODBCO", 0);
financeiro.setCampo("CODTIPTIT", 2);
```

```
  financeiro.setCampo("DTVENC", "04/10/2012");
  financeiro.setCampo("HISTORICO", "REEMBOLSO DE KM PARA O VEÍCULO " +
  getParam("CODVEICULO"));
```

Quando o "save" do registro é acionado, a alteração no Banco de dados é realizada. Portanto aqui estamos incluindo um registro na TGFFIN. Observe:

```
  //Quando o "save" do registro é acionado,
  //a alteração é feita no Banco de dados.

  financeiro.save();
```

Finalmente configuramos uma mensagem para ser exibida após a execução da ação.

```
  mensagem = "Foi gerado o título ";
  mensagem += financeiro.getCampo("NUFIN");
  mensagem += " no valor de ";
  mensagem += financeiro.getCampo("VLRDESDOB")
  mensagem += " como reembolso de KM para o veículo ";
  mensagem += getParam("CODVEICULO");
```

Texto

```
mensagem = "Foi gerado o título ";
mensagem += financeiro.getCampo("NUFIN");
mensagem += " no valor de ";
mensagem += financeiro.getCampo("VLRDESDOB")
mensagem += " como reembolso de KM para o veículo ";
mensagem += getParam("CODVEICULO");
```

Crie um parâmetro do tipo Pesquisa apontando para a tabela de veículos, em que você possa determinar para qual veículo será lançado o reembolso.

Assim, ao ser acionada, nossa rotina irá criar um título financeiro com o somatório de todos os lançamentos para o veículo informado.

Como tirar dúvidas

Para tirar dúvidas e compartilhar informações, use a sala Botões de Ação da comunidade Sankhya Developer.

Updated over 2 years ago

Próxima página

Rotina Lançador →

Rotina Banco de dados →

Rotina Java →