DAO - Camada de abstração de acesso a dados

A camada DAO no Miolo tem por objetivo fazer a abstração do acesso a bancos de dados relacionais, encapsulando a camada de acesso fornecida pelo PHP, permitindo usar uma única interface de programação, independente do banco de dados sendo utilizado. Embora existam várias soluções para este problema, com frameworks específicos (tais como o AdoDB e as classes PEAR), o Miolo tem implementado sua própria versão de DAO, que é descrita neste documento.
A implementação atual tem as seguintes características:

- O mecanismo de acesso a dados fornecido pelo PHP é encapsulado, permitindo uma interface única de programação.
- São fornecidos mecanismos básicos para geração automática de código SQL adaptado ao banco (inclusive joins, offsets e número máximo de linhas retornadas), uso de geradores (sequences) e conversão de datas e horários para um formato padrão.
- Uso de transações, utilizando recursos nativos do banco de dados sendo acessado.
- Abstração de resultados de consultas (queries) em ResultSets, permitindo operações como travessia (browse), paginação, filtragem e ordenação do resultado de uma consulta.

A figura 1 fornece uma visão geral da estrutura de classes da camada DAO. As classes Connection, Query, Transaction, IdGenerator e SQLJoin são especializadas para cada banco de dados (PostgreSql, MySQL, Oracle, etc) a fim de implementar as particularidades de sintaxe e encapsular as funções PHP de acesso aos dados.

Configuração

As bases de dados que serão acessadas pelos módulos do MIOLO são configuradas no arquivo XML <miolo>/etc/miolo.conf. As entradas neste arquivo seguem a seguinte sintaxe:

```
<db>
   <base_name>
       <system>dbms</system>
       <host>address</host>
       <name>db_name</name>
       <user>username</user>
       <password>password</password>
   </base_name>
</db>
```

<base_name>: nome da base de dados, conforme ela será referenciada pelos módulos do MIOLO. Um mesmo banco de dados pode ser referenciado por nomes diferentes, para implementar restrições de segurança, por exemplo.
<dbms>: o Sistema Gerenciador de Banco de Dados. O MIOLO implementa drivers para: postgres, mysql, oracle, firebird, sqlite, mssql
<address>: endereço do servidor executando o DBMS (geralmente um endereço IP).
<db_name>: nome do banco de dados, conforme referenciado pelo DBMS, ou pelo software cliente.
<username>: identificação do usuário para acesso ao banco de dados.
<password>: senha do usuário para acesso ao banco de dados.

Classes DAO

Embora constituída por várias classes, o usuário necessita de acesso a apenas alguns poucos métodos oferecidos pela camada DAO. Nesta seção apresentamos a definição dos principais métodos usados.
Alguns conceitos são importantes para compreensão do funcionamento da camada DAO:

Campo: corresponde a uma coluna de uma tabela (ou visão) no banco de dados.
Registro: corresponde a uma linha de uma tabela (ou visão) no banco de dados.
ResultSet: corresponde a um array bidimensional, retornado como resultado de uma consulta (query) SQL. Cada linha corresponde a um registro e cada coluna corresponde a um campo. As colunas têm índice numérico a partir de 0, e são ordenadas de acordo com o comando SQL SELECT.
DataSet: corresponde a uma instância da classe DataSet, usada internamente pela camada de acesso a dados, para encapsular o acesso a um ResultSet.

O procedimento básico para acesso a uma base de dados tem a seguinte seqüência de ações:

1. Instanciar um objeto Database (via $MIOLO->GetDatabase), que encapsula a conexão e o acesso ao banco de dados;
2. Instanciar um objeto Sql, que encapsula um comando SQL;

3. Realizar a consulta, criando um objeto Query, ou executar o(s) comando(s) SQL;
4. Trabalhar com o ResultSet, através do objeto Query.

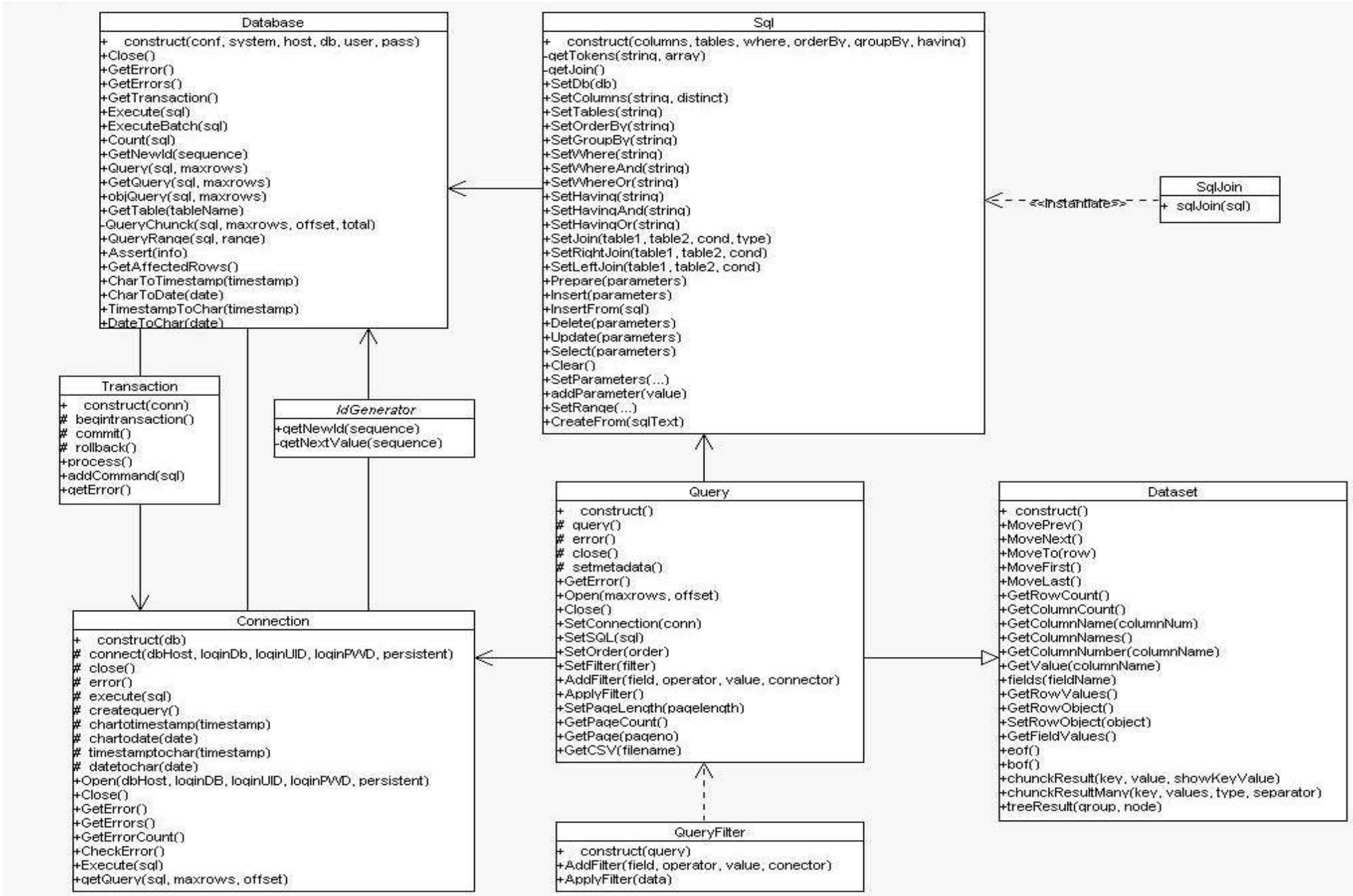

Figura 1 – Visão geral das classes DAO

classe Database

function __contruct($conf,$system,$host,$db,$user,$pass)

Abre uma conexão com o banco de dados, criando uma instancia do objeto Connection correspondente àquele banco. Os parâmetros para a conexão são os descritos no arquivo de configuração miolo.conf; $conf é o nome da base de dados. Este método é chamado geralmente através de $MIOLO->GetDatabase($conf).

function Close()

Fecha a conexão com o banco de dados.

function GetError()

Obtém o primeiro erro da lista de erros da conexão, se existir.

function GetErrors()

Obtém a lista de erros relativa à conexão.

function GetTransaction()

Obtém um objeto Transaction, para controle de uma transação. Geralmente é usado internamente, pelo método ExecuteBatch.

function Execute($sql)

Executa o comando $sql.

function ExecuteBatch($sql)

Executa os comandos SQL de $sql. $SQL é geralmente um array de comandos, simulando a execução de um script, dentro do escopo de uma transação.

function Count($sql)

Retorna o número de registros do resultado da execução de $sql.

function GetNewId($sequence)

Retorna o próximo valor da seqüência $sequence (cadastrada no banco de dados). Usado na geração automática de OID e chaves primárias para as tabelas.

function Query($sql,$maxrows=0) [deprecated]

Retorna o ResultSet correspondente à execução de $sql, com no máximo $maxrows linhas.

function GetQuery($sql,$maxrows=0)

Retorna um objeto Query, correspondente à execução do comando definido pelo objeto $sql.

function objQuery($sql,$maxrows=0) [deprecated]

Retorna um objeto Query, correspondente à execução do comando definido pelo comando SQL SELECT $sql.

function GetTable($tablename)

Retorna um objeto Query, com todos os registros e campos da tabela $tablename. Corresponde à execução de “select * from $tablename”.

function QueryRange($sql,&$range)

Retorna o ResultSet correspondente à execução do comando $sql. $range é uma instância da classe QueryRange e delimita a faixa de registros retornados.

private function QueryChunk($sql, $maxrows, $offset, &$total)

Retorna o objeto Query correspondente à execução do comando $sql. $maxrows indica o máximo de registros a serem retornados, $offset indica o deslocamento a partir do início da tabela e $total retorna o efetivo número de registros do resultado.

function Assert($info=false)

Interrompe a execução do script caso haja um erro no acesso ao banco de dados.

function GetAffectedRows()

Retorna o número de registros afetados pela execução do último comando SQL.

function CharToTimestamp($timestamp)
Faz a conversão do tipo char para timestamp (dependente do banco de dados).

function CharToDate($date)
Faz a conversão do tipo char para date (dependente do banco de dados).

function TimestampToChar($timestamp)
Faz a conversão do tipo timestamp para char (dependente do banco de dados).

function DateToChar($date)
Faz a conversão do tipo date para char (dependente do banco de dados).

classe Dataset

A classe Dataset é usada internamente pela camada de acesso a dados, definindo métodos para acesso aos ResultSet originados pela execução de consultas SQL. As consultas são realizadas através de objetos da classe Query (que estende a classe Dataset). Os métodos descritos aqui são acessados através de um objeto Query, retornado por um método da classe Database. É usado o conceito de "ponteiro" para indicar a posição do ResultSet está sendo acessada em determinado momento e de "registro corrente" para indicar a linha referenciada pelo "ponteiro".

function MovePrev()
Move o ponteiro para o registro anterior ao corrente (ou BOF).

function MoveNext()
Move o ponteiro para o registro seguinte ao corrente (ou EOF).

function MoveFirst()
Move o ponteiro para o primeiro registro do ResultSet.

function MoveLast()
Move o ponteiro para o último registro do ResultSet.

function MoveTo($row)
Move o ponteiro para o registro $row do ResultSet.

function GetRowCount()
Retorna quantas linhas o ResultSet possui.

function GetColumnCount()
Retorna quantas colunas o ResultSet possui.

function GetColumnName($columnNum)
Retorna o nome do campo correspondente à coluna $columnNum.

function GetColumnNames()
Retorna um array com os nomes de todos os campos.

function GetColumnNumber($columnName)
Retorna o número da coluna correspondente ao campo com nome $columnName.

function GetValue($columnName)
Retorna o conteúdo do campo $columnName no registro corrente.

function fields($fieldName)
Retorna o conteúdo do campo $fieldName no registro corrente.

function GetRowValues()
Retorna a linha correspondente ao registro corrente, como um array com índices numéricos.

function GetFieldValues()
Retorna a linha correspondente ao registro corrente, como um array com índices alfabéticos (nome dos campos).

function GetRowObject()

Retorna um objeto cujas propriedades correspondem, cada uma, a um campo do registro corrente. O nome da propriedade é o nome do campo e o valor da propriedade é o valor do campo no registro corrente.

function SetRowObject($object)
Cria (ou atribui) propriedades ao objeto $object, onde cada propriedade corresponde a um campo do registro corrente. O nome da propriedade é o nome do campo e o valor da propriedade é o valor do campo no registro corrente.

function eof()
Indica (true/false) se foi alcançado o fim do ResultSet.

function bof()
Indica (true/false) se foi alcançado o início do ResultSet.

function chunkResult($key=0, $value=1, $showKeyValue=true)
Faz a compactação do ResultSet em um array associativo, cujos índices são determinados pelos valores da coluna $key e cujos valores são determinados pelos valores da coluna $value. $showKeyValue indica se o valor do índice será exibido no conteúdo do array.

function chunkResultMany($key, $values, $type='S', $separator='')
Faz a compactação do ResultSet em um array associativo, cujos índices são determinados pelos valores da coluna $key. $values é um array indicando quais colunas serão agregadas. $type determina se os valores serão agrupados como strings ($type='S'), separados por $separator, ou se serão colocados em um array.

function treeResult($group, $node)
Cria uma estrutura em árvore, baseada em um array associativo. $group indica quais campos devem ser agrupados, e $node indica quais campos formarão o valor do nó.

classe Query

A classe Query encapsula os métodos para a execução de consultas ao banco de dados. Esta classe define alguns métodos abstratos que são implementados pelas classes internas para cada banco de dados específico. Os métodos de acesso ao ResultSet são herdados da classe Dataset.

protected function _query()
Realiza efetivamente o acesso ao banco de dados.

protected function _error()
Indica se houve algum erro no acesso ao banco de dados.

protected function _close()
Libera os recursos usados na consulta.

protected function _setmetadata()
Obtém metadados referentes à consulta (p.ex. nome e tipo dos campos).

function GetError()
Retorna o erro encontrado na consulta, se houver.

function Open($maxrows=null, $offset=null)
Executa a consulta, retornando no máximo $maxrows a partir do registro $offset.

function Close()
Libera os recusos usados na consulta.

function SetConnection(&$c)
Indica qual conexão será usada para consulta. $c é um objeto Connection.

function SetSQL($sql)
Indica o comando SQL, através do objeto $sql, correspondente à consulta.

function SetOrder($order)
Ordena as colunas do ResultSet, segundo a ordem especificada por $order. $order é uma string com os nomes dos campos separados por ",".

function isFiltered()

Retorna se o ResultSet está filtrado ou não (true/false).

function AddFilter($field, $operator, $value, $connector)

Adiciona um filtro ao ResultSet. Um filtro é uma operação de comparação de um campo com um valor. $field é o nome do campo, $operator é a operação de comparação (“=”, “!=”,”like” ou “regex” – para expressões regulares). No caso da operação “like”, os caracteres curinga ‘?’ e ‘_’ substituem um caracter e o caracter ‘%’ substitui um conjunto de caracteres na expressão $value. $connector é usado para conectar os vários filtros e pode assumir os valores ‘AND’ (default) ou ‘OR’.

function ApplyFilter()

Aplica os filtros no ResultSet (se houver algum).

function SetPageLength($pagelength)

Indica o tamanho da página (número de registros) quando o ResultSet for paginado (geralmente em operações com o controle DataGrid).

function GetPageCount()

Quando o ResultSet for paginado, retorna o número de páginas.

function GetPage($page)

Quando o ResultSet for paginado, retorna um subconjunto do ResultSet, correspondente à página $page. A numeração das páginas começa em 1.

function GetCSV($filename = '')

Converte o ResultSet para o formato CSV.

classe Sql

A classe Sql é usada para encapsular um comando SQL, com seus respectivos parâmetros. No comando SQL o uso de parâmetros é indicado pelo caracter “?”. Antes da execução do comando, os parâmetros devem ser substituídos por seus valores atuais.

function __construct ($columns='',$tables='',$where='',$orderBy='',$groupBy='',$having='')

Instancia um objeto Sql. Cada um dos parâmetros, quando fornecido, é usado na construção do comando SQL. Nas consultas realizadas através de joins, o parâmetro $tables deve ser omitido, e devem ser usados os métodos relativos a joins.

function SetDb($db)

Indica o objeto Database usado para acesso ao banco de dados.

function SetColumns($string, $distinct=false)

Indica as colunas para o comando SQL. $distinct indica se será usada ou não a cláusula DISTINCT.

function SetTables($string)

Indica ao tabelas para o comando SQL.

function SetOrderBy($string)

Indica a ordenação das colunas no comando SQL SELECT.

function SetGroupBy($string)

Indica a chave de agrupamento no comando SQL SELECT.

function SetWhere($string)

Indica a expressão para a cláusula WHERE no comando SQL.

function SetWhereAnd($string)

Indica uma expressão a ser acrescentada à clausula WHERE via conector ‘AND’.

function SetWhereOr($string)

Indica uma expressão a ser acrescentada à clausula WHERE via conector ‘OR’.

function SetHaving($string)

Indica a expressão para a cláusula HAVING no comando SQL SELECT.

function SetHavingAnd($string)
Indica uma expressão a ser acrescentada à clausula HAVING via conector 'AND'.

function SetHavingOr($string)
Indica uma expressão a ser acrescentada à clausula HAVING via conector 'OR'.

function SetJoin($$table1, $table2, $condition, $type='INNER')
Indica que a consulta será realizada através do join das tabelas $table1 e $table2 usando a expressão $condition. $type pode ser 'INNER' (default), 'RIGHT' ou 'LEFT'.

function SetRightJoin($table1, $table2, $condition)
Indica que a consulta será realizada através de right join das tabelas $table1 e $table2 usando a expressão $condition.

function SetLeftJoin($table1, $table2, $condition)
Indica que a consulta será realizada através de left join das tabelas $table1 e $table2 usando a expressão $condition.

function Prepare($parameters)
Faz a substituição dos parâmetros formais pelos valores dos parâmetros atuais. $parameters é um array com os valores
dos parâmetros (ou um valor, caso o parâmetro seja único).

function Insert($parameters=null)
Gera uma instrução SQL INSERT, fazendo a substituição dos parâmetros formais pelos valores dos parâmetros atuais, se necessário.

function InsertFrom($sql)
Gera uma instrução SQL INSERT, utilizando como fonte de dados uma consulta SQL expressa através da string $SQL.

function Delete($parameters=null)
Gera uma instrução SQL DELETE, fazendo a substituição dos parâmetros formais pelos valores dos parâmetros atuais, se necessário.

function Update($parameters=null)
Gera uma instrução SQL UPDATE, fazendo a substituição dos parâmetros formais pelos valores dos parâmetros atuais, se necessário.

function Select($parameters=null)
Gera uma instrução SQL SELECT, fazendo a substituição dos parâmetros formais pelos valores dos parâmetros atuais, se necessário.

function Clear()
Remove o conteúdo dos atributos, inicializando o objeto.

function SetParameters($parameter | $param1, $param2, …)
Indica o atributo parameters. $parameter pode ser um valor único, um array de valores ou ainda vários valores passados como argumentos para o método.

function addParameters($value)
Acrescenta o valor $value no array de parâmetros do comando SQL.

function SetRange($range | $page, $rows)
Indica o atributo range para a consulta SQL SELECT. $range é um objeto do tipo QueryRange. $page indica uma página e $rows indica o número máximo de registros a retornar da consulta.

function CreateFrom($sqltext)
Estabelece os atributos do objeto Sql com base um comando texto SQL SELECT.

Exemplos

Para efeito dos exemplos, será usada a base de dados "common", as tabelas cm_acesso, cm_transacao e cm_sistema e a sequence seq_cm_transacao. Estas tabelas têm a seguinte estrutura:

cm_acesso (idgrupo (PK), idtrans(PK), direito)
cm_transacao (idtrans (PK), transacao, idsistema (FK))
cm_sistema (idsistema (PK), sistema)

Exibir todos os campos de todos os registros

Código:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('common'); // instancia um objeto Database
$sql = new sql('*','cm_transacao'); // instancia um objeto SQL com os dados para a consulta
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
$n = $query->GetRowCount(); // obtem o número de registros retornados
$result = $query->result; // trabalha com o ResultSet
for ($i=0; $i < $n; $i++)
{
   echo "#$i - " . $result[$i][1] . '<br>'; // acessa o campo ‘transacao’
}
```

Consulta com critério de seleção (WHERE)

Código:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('common'); // instancia um objeto Database
$sql = new sql('*','cm_transacao', ‘idtrans = 100’); // instancia um objeto SQL
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
$n = $query->GetColumnCount(); // obtem o número de colunas
$result = $query->result; // trabalha com o ResultSet
for ($i=0; $i < $n; $i++)
{
   echo $result[0][$i] . '<br>'; // acessa, no primeiro registro, cada campo retornado
}
```

Consulta com critério de seleção composto

Código:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('common'); // instancia um objeto Database
$sql = new sql('*','cm_transacao', “(idtrans = 100) AND (transacao LIKE ‘S%’”);
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
```

Código alternativo:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('common'); // instancia um objeto Database
$sql = new sql('*','cm_transacao'); // instancia um objeto SQL
$SQL->SetWhere(“(idtrans = 100) AND (transacao LIKE ‘S%’”);
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
```

Código alternativo:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('common'); // instancia um objeto Database
$sql = new sql('*','cm_transacao'); // instancia um objeto SQL
$SQL->SetWhere(‘idtrans = 100’);
$SQL->SetWhereAnd(“transacao LIKE ‘S%’”);
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
```

Consulta com parâmetro único

Código:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('common'); // instancia um objeto Database
$sql = new sql('*','cm_transacao', ‘idtrans = ?’); // instancia um objeto SQL
$sql->SetParameters(100); // indica o valor do parâmetro
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
```

Consulta com múltiplos parâmetros

Código:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('common'); // instancia um objeto Database
$sql = new sql('*','cm_transacao', “(idtrans = ?) AND (transacao LIKE ?)”);
$sql->SetParameters(100, ‘A%’); // indica os valores dos parâmetros
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
```

Código alternativo:

```
global $MIOLO;  // acesso a clase base do framework
$db = $MIOLO->GetDatabase('common'); // instancia um objeto Database
$sql = new sql('*','cm_transacao', “(idtrans = ?) AND (transacao LIKE ?)”);
$sql->SetParameters(array(100, ‘A%’)); // indica os valores dos parâmetros como array
$query = $db->GetQuery($sql); // executa a consulta e obtem um objeto Query
```

Consulta com ordenação

Código:
```
	$db = $MIOLO->GetDatabase('common');
	$sql = new sql('*','cm_transacao','','transacao');
	$query = $db->GetQuery($sql);
```

Consulta com inner join entre 2 tabelas e uso de aliases
Código:
```
	$db = $MIOLO->GetDatabase('common');
	$sql = new sql('s.sistema, t.transacao','','','s.sistema,t.transacao');
	$sql->SetJoin('cm_sistema s','cm_transacao t','(s.idsistema=t.idsistema)');
	$query = $db->GetQuery($sql);
```

Consulta com left join entre 3 tabelas, uso de aliases, função de grupo e cláusula GROUP BY
Código:
```
	$db = $MIOLO->GetDatabase('common');
	$sql = new sql('s.sistema, t.transacao, count(a.idgrupo)','','', 's.sistema,t.transacao',
's.sistema,t.transacao');
	$sql->SetLeftJoin('cm_sistema s','cm_transacao t','(s.idsistema=t.idsistema)');
	$sql->SetLeftJoin('cm_transacao t','cm_acesso a','(t.idtrans=a.idtrans)');
	$query = $db->GetQuery($sql);
```

Consulta com uso da cláusula HAVING
Código:
```
	$db = $MIOLO->GetDatabase('common');
	$sql = new sql('s.sistema, count(t.idtrans)','','','s.sistema','s.sistema', '(count
(t.idtrans) > 3)');
	$sql->SetLeftJoin('cm_sistema s','cm_transacao t','(s.idsistema=t.idsistema)');
	$query = $db->GetQuery($sql);
```

Consulta com subquery
Código:
```
	$db = $MIOLO->GetDatabase('common');
	$sql = new sql('a.idgrupo','cm_acesso a');
	$sql->SetWhere('a.idtrans IN ?');
	$sqlx = new sql('t.idtrans','cm_transacao t','(t.idsistema = 1)'); // Sql para a subquery
	$sql->SetParameters(':(' . $sqlx->Select() . ')'); // gera o SQL SELECT como parâmetro
	$query = $db->GetQuery($sql);
```

Observar neste exemplo o uso do caracter ‘:’ antes do valor do parâmetro. O uso do ‘:’ impede que sejam acrescentadas as aspas ao valor do parâmetro.

Acessando uma linha específica
Código:
```
	$db = $MIOLO->GetDatabase('common');
	$sql = new sql('*','cm_transacao');
	$query = $db->GetQuery($sql);
	$n = $query->GetRowCount();
	$query->MoveTo(4);
```

Navegação (browse) no ResultSet
Código:
```
	$db = $MIOLO->GetDatabase('common');
	$sql = new sql('*','cm_transacao');
	$query = $db->GetQuery($sql);
	echo 'Move to First: <br>';
//
// acessa o primeiro registro
//
	$query->MoveFirst();
	$row = $query->GetRowValues();
	foreach($row as $f)
	{
	   echo $f . ' - ';
	}
	echo '<br><br>';
//
// acessa o último registro
//
	echo 'Move to Last: <br>';
	$query->MoveLast();
	$row = $query->GetRowValues();
	foreach($row as $f)
```

```
       {
          $text .= $f . ' - ';
       }
       echo '<br><br>';
//
// browse
//
       echo 'Transverse: <br>';
       $query->MoveFirst();
       while (!$query->eof())
       {
          echo $query->fields('transacao') . '<br>';
          $query->MoveNext();
       }
```

Contando registros
Código:
```
       $db = $MIOLO->GetDatabase('common');
       $sql = new sql('*','cm_transacao');
       $n = $db->Count($sql);
       $command = $sql->Select();
       $echo "$command => #records: " . $n . '<br><br>';
       $sql = new sql('*','cm_transacao','(idtrans > 100)');
       $n = $db->Count($sql);
       $echo "$command => #records: " . $n . '<br><br>';
```

Usando range para paginação
Código (com objeto QueryRange):
```
       $db = $MIOLO->GetDatabase('common');
       $sql = new sql('*','cm_transacao');
       $n = $db->Count($sql);  // total de registros
       $totalpages = (int) $n / 5;  // cada página com 5 registros
       for($page=1; $page <= $totalpages; $page++)
       {
         $range = new QueryRange($page,5);
         $result = $db->QueryRange($sql->Select(), &$range);
         $n = $range->total;
         echo "Page: $page  [$n records]: <br>";
         echo '          #rec - transação' . '<br>';
         for ($i=0; $i < $n; $i++)
         {
            echo "         #$i - " . $result[$i][1] . '<br>';
         }
         echo "<br>";
       }
```

Código (com objeto Sql):
```
       $db = $MIOLO->GetDatabase('common');
       $sql = new sql('*','cm_transacao');
       $n = $db->Count($sql); // total de registros
       $totalpages = (int) $n / 5; // cada página com 5 registros
       for($page=1; $page <= $totalpages; $page++)
       {
         $sql->SetRange($page,5);
         $query = $db->GetQuery($sql);
         $result = $query->result;
         $n = $query->GetRowCount();
         echo "Page: $page  [$n records]: <br>";
         echo '          #rec - transação' . '<br>';
         for ($i=0; $i < $n; $i++)
         {
            echo "         #$i - " . $result[$i][1] . '<br>';
         }
         echo "<br>";
       }
```

Paginação com objeto Query
Código:
```
       $db = $MIOLO->GetDatabase('common');
       $sql = new sql('*','cm_transacao');
       $query = $db->GetQuery($sql);
       $query->SetPageLength(10);  // indica o tamanho de cada página
       $n = $query->GetPageCount(); // obtém a quantidade páginas
       echo "<b>$n pages</b><br><br>";
```

```
       for($page=1; $page <= $n; $page++)
       {
           $result = $query->GetPage($page); // subconjunto do ResultSet, correspondente a pagina
           $m = count($result); // a pagina pode não estar completa
           echo "Page: $page  [$m records]: <br>";
           echo '          #rec - transação' . '<br>';
           for ($i=0; $i < $m; $i++)
           {
              echo "         #$i - " . $result[$i][1] . '<br>';
           }
           echo "<br>";
       }
```

Geração automática de ID (sequences)
Código:

```
       $db = $MIOLO->GetDatabase('common');
       $id = $db->GetNewId('seq_cm_transacao');
```

Transações
Código:

```
       $db = $MIOLO->GetDatabase('common');
//
// Exemplo com Commit
//
       $id = $db->GetNewId('seq_cm_transacao');  // inserir nova transacao ‘Teste’
       $sql = new sql('idtrans, transacao, idsistema','cm_transacao');
       $args = array($id,'Teste',1);
       $cmd = array(); // array com os comandos SQL
       $cmd[] = $sql->Insert($args);
       $sql->sql('idtrans, idgrupo, direito','cm_acesso'); // reconstrói o objeto Sql
       $cmd[] = $sql->Insert(array($id, 1, 1));
       $cmd[] = $sql->Insert(array($id, 2, 1));
       $cmd[] = $sql->Insert(array($id, 3, 1));
       $cmd[] = $sql->Insert(array($id, 4, 1));
       $cmd[] = $sql->Insert(array($id, 5, 1));
       $ok = $db->Execute($cmd); // como $cmd é array, vai gerar uma transação
       echo "id = $id <br>";
       echo ($ok ? 'Transaction Ok' : 'Transaction Fail');
       echo '<br>';
//
// Exemplo com RollBack
//
       $id = $db->GetNewId('seq_cm_transacao');
       $sql = new sql('idtrans, transacao, idsistema','cm_transacao');
       $args = array($id,'Teste2',1);
       $cmd = array();
       $cmd[] = $sql->Insert($args);
       $sql->sql('idtrans, idgrupo, direito','cm_acesso');
       $cmd[] = $sql->Insert(array($id, 1, 1));
       $cmd[] = $sql->Insert(array($id, 2, 1));
       $cmd[] = $sql->Insert(array('aaa', 3, 1)); // este comando gerará um erro!
       $cmd[] = $sql->Insert(array($id, 4, 1));
       $cmd[] = $sql->Insert(array($id, 5, 1));
       $ok = $db->Execute($cmd);
       echo "id = $id <br>";
       echo ($ok ? 'Transaction Ok' : 'Transaction Fail ' . $db->GetError());
```

Ordenando o resultset
É possível ordenar o resultado de uma consulta (resultset). Não se trata de indicar uma cláusula ORDER BY para um comando SQL SELECT, mas sim ordenar o array resultante de uma consulta já realizada (usando funções do PHP).

Código:

```
       $db = $MIOLO->GetDatabase('common');
       $sql = new sql('*','cm_transacao',"transacao like 'A%'");
       $query = $db->GetQuery($sql);
       $n = $query->GetRowCount();
       echo 'Results - before order' . ‘<br>’;
       for ($i=0; $i < $n; $i++)
       {
          echo "         #$i - " . $result[$i][0] . ‘ – ‘ . $result[$i][1] . '<br>';
       }
```

```
echo "<br>";
$query->SetOrder('transacao');
echo 'Results - after order' . '<br>';
for ($i=0; $i < $n; $i++)
{
   echo "         #$i - " . $result[$i][0] . ' – ' . $result[$i][1] . '<br>';
}
echo "<br>";
```

Filtrando o resultset

É possível filtrar o resultado de uma consulta (resultset). Não se trata de indicar uma cláusula WHERE para um comando SQL SELECT, mas sim filtrar o array resultante de uma consulta já realizada. Como o filtro modifica o resultset, não é possível voltar ao resultset original depois de aplicado o filtro. Podem ser usados os operadores '=', '!=','like' e 'regex'.

Código com operador 'like':

```
$db = $MIOLO->GetDatabase('common');
$sql = new sql('*','cm_sistema');
$query = $db->GetQuery($sql);
$query->AddFilter('sistema','like','C%');
$query->ApplyFilter();
```

Código com operador 'regex':

```
$db = $MIOLO->GetDatabase('common');
$sql = new sql('*','cm_sistema');
$query = $db->GetQuery($sql);
$query->AddFilter('sistema','regex','^(.*?)A(.*)');
$query->ApplyFilter();
```