Query Builder - Codeigniter 3

$this->db->get()

Executa a seleção consulta e retorna o resultado. Pode ser usado por si para recuperar todos os registros de uma tabela:

```
$query = $this->db->get('minhaTabela');//SELECT * FROM minhaTabela
```

O segundo e terceiro parâmetros permitem definir um limite e cláusula de compensação:

```
$query = $this->db->get('minhaTabela', 10, 20); //SELECT * FROM minhaTabela LIMIT 20, 10
```

Você vai notar que a função acima é atribuído a uma variável chamada `$query`, que pode ser usado para mostrar os resultados:

```
$query = $this->db->get('minhaTabela');
foreach($query->result() as $linha){
    echo $linha->titulo;
}
```

$this->db->get_compiled_select()

Compila a seleção consulta apenas como `$this->db->get()` , mas não executar a consulta. Este método simplesmente retorna a consulta SQL como uma string:

```
$sql = $this->db->get_compiled_select('minhaTabela');
echo $sql; //"SELECT * FROM minhaTabela"
```

O segundo parâmetro permite definir ou não a consulta construtor de consulta será reposto (por padrão, ele será zerado, assim como quando usando `$this->db->get()`):

```
echo $this->db->limit(10, 20)->get_compiled_select('minhaTabela', FALSE); //"SELECT * FROM LIMIT
mytable 20, 10"

echo $this->db->select('titulo, conteudo, data')->get_compiled_select(); //"SELECT * FROM minhaTa
bela LIMIT 20, 10"
```

A principal coisa a notar no exemplo acima é que a segunda consulta não utilizar `$this->db->from()` e não passar um nome de tabela para o primeiro parâmetro. A razão para este resultado é porque a consulta não foi executada usando `$this->db->get()` que redefine valores ou redefinir directamente utilizando `$this->db->reset_query()` .

$this->db->get_where()

Idêntico à função acima, exceto que ele permite que você adicione uma cláusula “where” no segundo parâmetro, em vez de usar o db->where():

```
$query = $this->db->get_where('minhaTabela', array('id' => $id), $limit, $offset);
```

$this->db->select()

Permite-lhe para escrever a parte SELECT da sua consulta:

```
$this->db->select('titulo, conteudo, data');
$query = $this->db->get('minhaTabela');
//SELECT titulo, conteudo, data FROM minhaTabela
```

`$this->db->select()` aceita um segundo parâmetro opcional. Se você configurá-lo para FALSE, CodeIgniter não vai tentar proteger seu campo ou tabela nomes. Isso é útil se você precisar de uma instrução SELECT composto em que escape automático de campos pode quebrá-las.

```
$this->db->select('(SELECT SUM(pagamentos.quantidade) FROM pagamentos WHERE pagamentos.fatura_id=
4') AS quantidade_paga', FALSE);
```

$this->db->select_max()

Grava um Select MAX (campo) parte para a sua consulta . Você pode, opcionalmente, incluir um segundo parâmetro para renomear o campo resultante.

```
$this->db->select_max('idade');
$query = $this->db->get('membros');
// Seleciona membro com maior idade

$this->db->select_max('idade, membro_idade');
$query = $this->db->get('membros');
// Seleciona o membro com maior idade e irá renomeá-lo par membro_idade
```

$this->db->select_min()

Grava um “SELECT MIN (campo)” parte para a sua consulta . Tal como acontece com select_max (), você pode opcionalmente incluir um segundo parâmetro para renomear o campo resultante.

```
$this->db->select_min('idade');
$query = $this->db->get('membros');
// Seleciona membro com menor idade
```

$this->db->select_avg()

Grava um “SELECT AVG (campo)” parte para a sua consulta . Tal como acontece com select_max (), você pode opcionalmente incluir um segundo parâmetro para renomear o campo resultante.

```
$this->db->select_avg('idade');
$query = $this->db->get('membros');
// Retorna média das idades dos membros
```

$this->db->select_sum()

Grava um “SELECT SUM (campo)” parte para a sua consulta . Tal como acontece com select_max (), você pode opcionalmente incluir um segundo parâmetro para renomear o campo resultante.

```
$this->db->select_sum('idade');
$query = $this->db->get('membros');
// Retorna soma das idades dos membros
```

$this->db->from()

Permite-lhe para escrever a partir da porção de sua consulta :

```
$this->db->select('titulo, conteudo, data');
$this->db->from('minhaTabela');
$query = $this->db->get();
// "SELECT titulo, conteudo, data FROM minhaTabela"
```

$this->db->join()

Permite-lhe para escrever a parte JOIN de sua consulta:

```
$this->db->select('*');
$this->db->from('blogs');
$this->db->join('comentarios', 'comentario.id =  blogs.id');
$query = $this->db->get();
// SELECT * FROM blogs JOIN comentarios ON comentarios.id = blogs.id
```

Várias chamadas de função pode ser feito se você precisar de várias associações em uma consulta.

Se você precisa de um tipo específico de se juntar a você pode especificá-lo via o terceiro parâmetro da função. As opções são: esquerda, direita, externa, interna, externa esquerda e para a direita exterior.

```
$this->db->join('comentarios', 'comentarios.id =  blogs.id', 'left');
// LEFT JOIN comentarios ON comentarios.id = blogs.id
```

$this->db->where()

Esta função permite que você defina ONDE cláusulas usando um dos quatro métodos:

```
$this->db->where('nome', $nome);
// WHERE nome = 'Aleck Yann'
```

Observe que o sinal de igual é adicionado para você.

Se você usar várias chamadas de função que eles vão ser encadeados com AND entre eles:

```php
$this->db->where('nome', $nome);
$this->db->where('title', $titulo);
$this->db->where('status', $status);
// WHERE nome = 'Aleck Yann' AND titulo = 'MIT' AND status = 'ativo'
```

Pode incluir um operador, em primeiro parâmetro, a fim de controlar a comparação:

```php
$this->db->where('nome =!', $nome);
$this->db->where('idade <', $idade);
// WHERE nome != 'Aleck Yann' AND idade < 22
```

> Todos os valores passados para esta função são escapados automaticamente, produzindo querys mais seguras.

$this_db_or_where()

Esta função é idêntica à descrita acima, excepto que as várias instâncias são unidas por OR:

```php
$this->db->where('nome =!', $nome);
$this->db->or_where('idade >', $idade);
// WHERE nome != 'Aleck Yann' OR idade > 22
```

$this->db->where_in()

Gera um campo WHERE IN (‘item’, ‘item’) consulta SQL juntou-se com e, se necessário:

```php
$nomes = array('Aleck Yann', 'Jonas', 'José');
$this->db->where_in('usuarios', $momes);
// WHERE username IN ('Frank', 'Todd', 'James')
```

$this->db->where_not_in()

Gera um campo WHERE IN (‘item’, ‘item’) SQL consulta unidos com ou se for o caso:

```php
$nomes = array('Aleck Yann', 'Jonas', 'José');
$this->db->where_not_in('usuarios', $nomes);
// OR usuarios IN ('Aleck Yann', 'Jonas', 'José')
```

$this->db->or_where_not_in()

Gera um campo WHERE NOT IN (‘item’, ‘item’) SQL consulta unidos com ou se for o caso:

```php
$nomes = array('Aleck Yann', 'Jonas', 'José');
$this->db->or_where_not_in('usuarios', $nomes);
// OR usuarios NOT IN ('Aleck Yann', 'Jonas', 'José')
```

$this->db->like()

Este método permite gerar cláusulas LIKE, úteis para fazer buscas.

> Todos os valores passados para este método são escapados automaticamente.

```
$this->db->like('titulo', 'saida');
// WHERE titulo LIKE %saida% ESCAPE '!'
```

Se você usar vários chamadas de método que vai ser encadeados com AND entre eles:

```
$this->db->like('titulo', 'saida');
$this->db->like('texto', 'saida');
//WHERE titulo LIKE %saida% ESCAPE '!' AND texto LIKE %saida% ESCAPE '!'
```

$this->db->or_like()

Este método é idêntico ao descrito acima, excepto que as várias instâncias são unidas por OR:

```
$this->db->or_like ('corpo', $match );
//OR corpo LIKE %match% ESCAPE '!'
```

$this->db->not_like()

Este método é idêntico ao not_like () , exceto que as várias instâncias são unidas por OR:

```
$this->db->not_like( 'titulo' , 'match' );
// WHERE titulo NOT LIKE '%match% ESCAPE '!'
```

$this->db->or_not_like()

Este método é idêntico ao not_like () , exceto que as várias instâncias são unidas por OR:

```
$this->db->or_not_like( 'texto' , 'match' );
// OR texto NOT LIKE '%match%' ESCAPE '!'
```

$this->db->group_by()

Permite-lhe para escrever o GROUP BY parte da sua consulta :

```
$this->db->group_by( "titulo" );
// GROUP BY titulo
```

$this->db->distinct()

Adiciona a palavra-chave “DISTINCT” a um consulta:

```
$this->db->distinct();
```

```php
$this->db->get( 'minhaTabela' );
// SELECT DISTINCT * FROM minhaTabela
```

$this->db->having()

Permite-lhe para escrever a parte que tem de sua consulta . Há 2 possíveis sintaxes, um argumento ou 2:

```php
$this->db->having( 'usuario_id = 45' );
// HAVING usuario_id = 45
```

$this->db->or_having()

Idêntico ao having(), apenas separa várias cláusulas com “OR”:

```php
$this->db->having( 'usuario_id = 45' );
// OR HAVING usuario_id = 45
```

$this->db->order_by()

Permite definir uma cláusula ORDER BY.
O primeiro parâmetro contém o nome da coluna que você gostaria de ordenar.
O segundo parâmetro permite definir a direção do resultado.
As opções são **ASC** , **DESC** E **RANDOM** .

```php
$this->db->order_by ( 'titulo' , 'DESC' );
// ORDER BY titulo DESC
```

Ou várias chamadas de função pode ser feito se você precisar de vários campos.

```php
$this->db->order_by( 'titulo' , 'DESC' );
$this->db->order_by( 'nome' ,  'ASC' );
// ORDER BY titulo DESC, nome ASC
```

Se você escolher o RANDOM opção de direção, então os primeiros parâmetros serão ignorados, a menos que você especificar um valor de semente numérico:

```php
$this->db->order_by('titulo' , 'RANDOM' );
// ORDER BY RAND()
$this->db->order_by( 42 , 'RANDOM' );
// ORDER BY RAND(42)
```

$this->db->limit()

Permite limitar o número de linhas que você gostaria devolvidos pela consulta :

```php
$this->db->limit( 10 );
// LIMIT 10
```

O segundo parâmetro permite definir um deslocamento resultado.

```
$this->db->limit( 10, 20 );
// LIMIT 20, 10
```

$this->db->count_all_results()

Permite que você para determinar o número de linhas em um Active Record em particular consulta . Consultas aceitará Consulta Construtor restritores tais como where () , or_where() , like() , or_like() , etc. Exemplo:

```
echo $this->db->count_all_results( 'minhaTabela' );
// 22
$this->db->like( 'titulo' , 'match' );
$this->db->from( 'minhaTabela' );
echo $this->db->count_all_results ();
// 17
```

$this->db->count_all()

Permite-lhe determinar o número de linhas em uma tabela particular. Apresentar o nome da tabela no primeiro parâmetro. Exemplo:

```
echo $this->db->count_all( 'minhaTabela' );
// 22
```

ESTILO DE AGRUPAMENTO

Consulta de agrupamento permite que você crie grupos de cláusulas WHERE quando colocadas entre parênteses. Isso permitirá que você para criar consultas com complexo de cláusulas WHERE. Grupos aninhados são suportados. Exemplo:

```
$this->db->select('*')->from('minhaTabela')
    ->group_start()
        ->where('a', 'a')
        ->or_group_start()
            ->where('b', 'b')
            ->where('c', 'c')
        ->group_end()
    ->group_end()
    ->where('d', 'd')
->get();

# SELECT * FROM (minhaTabela) WHERE ( a = 'a' OR ( b = 'b' AND c = 'c' ) ) AND d = 'd'
```

> grupos precisam ser equilibrados, certifique-se todos os group_start () é acompanhado por um group_end ().

this->db->group_start()

Inicia um novo grupo, adicionando um parêntese de abertura à cláusula WHERE da consulta .

$this->db->or_group_start()

Inicia um novo grupo, adicionando um parêntese de abertura à cláusula WHERE da consulta , prefixando-lo com ‘OR’.

$this->db->not_group_start()

Inicia um novo grupo, adicionando um parêntese de abertura à cláusula WHERE da consulta , prefixando-lo com ‘NÃO’.

$this->db->or_not_group_start()

Inicia um novo grupo, adicionando um parêntese de abertura à cláusula WHERE da consulta , antepondo-a com “ou não”.

$this->db->group_end()

Termina o grupo atual, adicionando um parêntese de fechamento à cláusula WHERE da consulta .

---

this->db->insert()

Gera uma seqüência de inserção com base nos dados que você fornecer, e executa o consulta . Você pode passar uma matriz ou um objeto para a função. Aqui está um exemplo usando uma matriz:

```
$dados = array ( 'titulo' => 'Um titulo' , 'nome' => 'Meu nome' , 'dados' => 'Meus dados' );
$this->db->insert('minhaTabela', $dados);
// INSERT INTO minhaTabela (titulo, nome, dados) VALUES ('Um titulo', 'Meu nome', 'Meus dados')
```

> Todos os valores são escapados automaticamente, produzindo querys mais seguras.

$this->db->get_compiled_insert()

Compila a inserção consulta apenas como $ this-> db-> insert (), mas não executar a consulta. Este método simplesmente retorna a consulta SQL como uma string.

```
$dados = array ( 'titulo' => 'Um titulo' , 'nome' => 'Meu nome' , 'dados' => 'Meus dados' );
$sql = $this->db->set($data)->get_compiled_insert( 'minhaTabela' );
echo $sql;
// INSERT INTO mytable (titulo, nome, dados) VALUES ('Um titulo', 'Meu nome', 'Meus dados');
```

$this->db->insert_batch()

Gera uma seqüência de inserção com base nos dados que você fornecer, e executa o consulta . Você pode passar uma matriz ou um objeto para a função. Aqui está um exemplo usando uma matriz:

```
$dados  =  array (
```

```
        array (
                'title'  =>  'O meu título' ,
                'name'  =>  'My Name' ,
                'date'  =>  'Minha data'
        ),
        array (
                'title'  =>  'Outro título' ,
                'name'  =>  'Outro nome' ,
                'date'  =>  'Outra data'
        )
);
$this->db->insert_batch('minhaTabela', $dados);
// INSERT INTO (titulo, nome, data) VALUES minha_tabela ( 'O meu título "," Meu nome', 'Minha dat
a'), ( 'Outro título', 'Outro nome', 'Outra data')
```

O primeiro parâmetro conterá o nome da tabela, o segundo é uma matriz associativa de valores.

> Todos os valores são escapados automaticamente, produzindo querys mais seguras.

$this->db->replace()

Esse método executa uma instrução REPLACE, que é basicamente o padrão SQL para (opcional) APAGAR + INSERT, usando PRIMÁRIAS e UNIQUE chaves como o fator determinante. No nosso caso, ela vai lhe poupar da necessidade de implementar lógicas complexas, com diferentes combinações de 'select ()' , 'update ()' , 'delete ()' e 'insert()' chama.

```
$dados  =  array(
        'title'  =>  'O meu título' ,
        'name'   =>  'My Name' ,
        'date'   =>  'Minha data'
);
$ this->db->replace('minhaTabela',  $dados;
// REPLACE INTO minhaTabela (title, name, date) VALUES ('My title', 'My name', 'My date')
```

No exemplo acima, se assumirmos que o título de campo é a nossa principal chave, em seguida, se uma linha contendo 'O meu título ", como o título de valor, essa linha será eliminado com os nossos novos dados da linha substituí-lo.

O uso do `set()` método também é permitido e todos os campos são automaticamente escapou, assim como com `insert ()` .

$this->db->set()

Esta função permite que você defina valores para inserções ou atualizações.

Ele pode ser usado em vez de passar uma matriz de dados diretamente para as funções de inserção ou de atualização:

```
$this->db->set('name',  $name );
$this->db->insert(minhTabela);
// INSERT INTO minhaTabela (name) VALUES ('{$name}')
```

Se você usar a função múltipla chamada eles serão montados corretamente com base em se você está

fazendo uma inserção ou uma atualização.

$this->db->update()

Gera uma seqüência de atualização e executa a consulta com base nos dados que você fornecer. Você pode passar uma matriz ou um objeto para a função. Aqui está um exemplo usando uma matriz:

```
$dados  =  array(
        'title'  =>  $titulo ,
        'name'  =>  $name ,
        'date'  =>  $date
);
$this->db->where( 'id' ,  $ id );
$this->db->update( 'minhaTabela' ,  $ dados );
// UPDATE minhaTabela SET title = '{$title}', name = '{$name}', date = '{$date}' WHERE id = $id
```

> Todos os valores são escapados automaticamente, produzindo querys mais seguras.

Você notará que o uso do this- função> db-> where () $, o que permite definir a cláusula WHERE. Você pode opcionalmente passar esta informação diretamente para a função de atualização como uma string:

```
$ this->db->update( 'minhaTabela' ,  $dados ,  "id = 4" );
```

Ou como uma matriz:

```
$this->db->update( 'minhaTabela' ,  $ dados ,  array( 'id'  =>  $ id ));
```

Você também pode usar a função `$this->db->set()` descrito acima ao realizar atualizações.

$this->db->update_batch()

Gera uma seqüência de atualização com base nos dados que você fornecer, e executa o consulta . Você pode passar uma matriz ou um objeto para a função. Aqui está um exemplo usando uma matriz:

```
$dados  =  array(
   array (
      'title'  =>  'O meu título'  ,
      'name'  =>  'My Name 2'  ,
      'date'  =>  'Minha data 2'
   ),
   array(
      'title'  =>  'Outro título '  ,
      ' name '  =>  ' Outro nome 2 '  ,
      ' date '  =>  ' Outra data 2 '
   )
);
$this->db->update_batch( 'mytable' ,  $dados ,  'title' );
# UPDATE minhaTabela SET name = CASE WHEN title = 'My title' THEN 'My Name 2' WHEN title = 'Anoth
er title' THEN 'Another Name 2'  ELSE name END,  date = CASE WHEN title = 'My title' THEN 'My dat
e 2' WHEN title = 'Another title' THEN 'Another date 2' ELSE date END WHERE title IN ('My title',
'Another title')
```

O primeiro parâmetro conterá o nome da tabela, o segundo é uma matriz associativa de valores, o terceiro parâmetro é a chave onde.

> Todos os valores são escapados automaticamente, produzindo querys mais seguras.

$this->db->get_compiled_update()

Isso funciona exatamente da mesma maneira como `$this->db->get_compiled_insert()` , exceto que ele produz uma seqüência UPDATE SQL em vez de uma cadeia de SQL INSERT.

Para obter mais documentação vista de informações para `$this->db->get_compiled_insert()` .