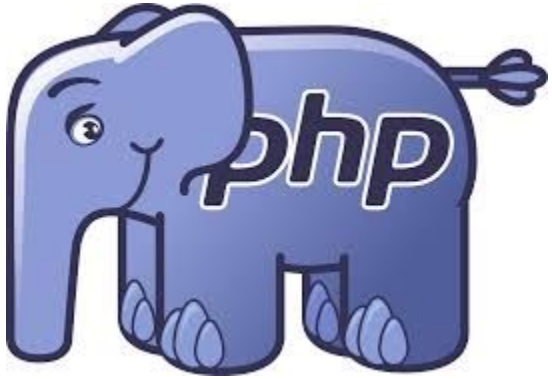
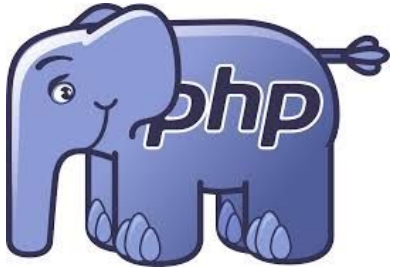
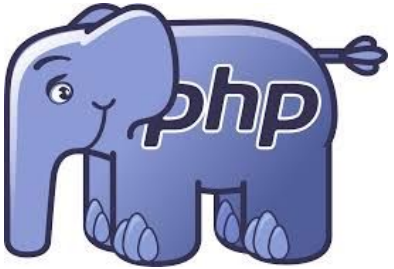


UpdateQuery com PHP 8.3



Comando para atualizar registro do banco de dados

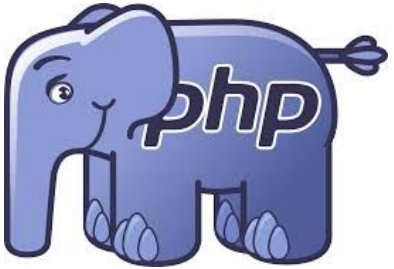
update aluno set nome = ‘WILTON’, cpf = ‘000.000.000-00’ where id = 1;



Query Builder

Um **query builder** é uma abstração que simplifica a construção de consultas SQL. Assim como utilizamos essa abstração para construir instruções `INSERT`, podemos usá-la da mesma maneira para criar comandos de atualização (`UPDATE`) em PHP.

Como utilizaremos a Classe InsertQuery?



Irá funcionar da seguinte maneira!

UpdateQuery::

table('aluno') → **Criar a classe abstrata e passa por parâmetro o nome da tabela**

->set($fieldAndValues) → **Passamos os valor e os campos para a classe**

->where('id', '=', $id) → **Executamos o filtro de dados com WHERE**

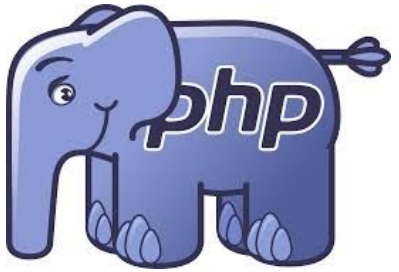
->update(); → **Executamos o script caso tudo seja executado recebemos true com retorno**

Aqui onde tudo acontece!


```php
$IsUpdate = UpdateQuery::table('ncm')
            ->set($fieldAndValues)
            ->where('ncm', '=', $value['codigo'])
            ->update();
```

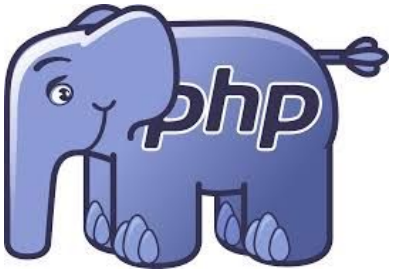
Caso o ‘UPDATE’ tenha executado corretamente, irá salvar as alterações no banco de dados, e retornará ‘TRUE’.

A Classe UpdateQuery



A classe UpdateQuery é uma *query builder* é utilizada para operações de atualização (update) em um banco de dados, facilitando a criação de consultas SQL de forma programática.

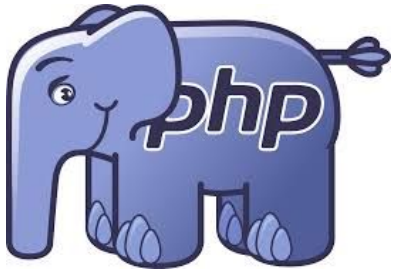
Estrutura Geral da Classe




```
<?php
# Namespace: A classe UpdateQuery está dentro do namespace
# app\database\builder. Isso ajuda a organizar o código,
# especialmente em projetos maiores, evitando conflitos de nome
# entre diferentes classes.

namespace app\database\builder;
```

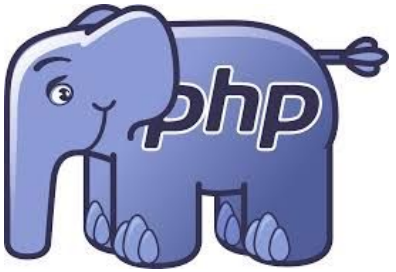
Estrutura Geral da Classe



 # Importação: Aqui, a classe Connection do namespace app\database
é importada para ser usada dentro da classe UpdateQuery.
Presume-se que a classe Connection gerencie a conexão com o
banco de dados.

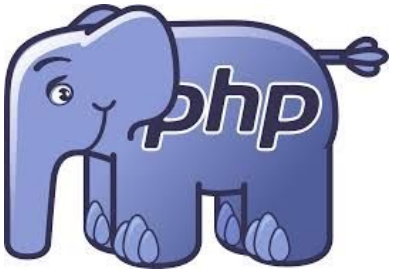
```
use app\database\Connection;
```

Propriedades da Classe



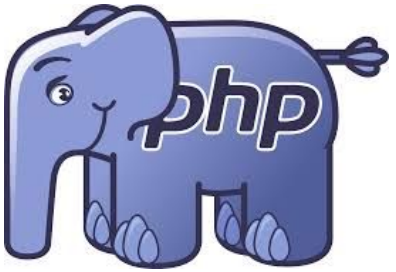

```
# $table: Armazena o nome da tabela onde a operação de update será realizada.
private string $table;
# $fieldsAndValues: Um array associativo onde as chaves representam os campos a
#serem  atualizados e os valores são os novos valores para esses campos.
private array $fieldsAndValues = [];
# $where: Um array que guarda as condições (cláusula WHERE) para a atualização.
private array $where = [];
# $binds: Armazena os valores que serão vinculados às placeholders na query para
evitar # injeções SQL.
private array $binds = [];
```

Método executeQuery



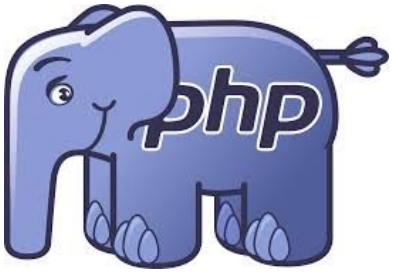

```
public function executeQuery($query)
{
        # Estabelece a conexão com o banco de dados.
        $connection = Connection::connection();
        # Prepara a query SQL para execução.
        $prepare      = $connection->prepare($query);
        # Executa a query preparada, passando os valores que
        # foram vinculados às placeholders.
        return $prepare->execute($this->binds ?? []);
}
```

Método where




```php
public function where(string $field, string $operator, string|int $value, ?string $logic = null)
{
    $placeHolder = '';
    $placeHolder = $field;
    if (str_contains($placeHolder, '.')) {
        $placeHolder = substr($field, strpos($field, '.') + 1);
    }
    $this->where[] = "{$field} {$operator} :{$placeHolder} {$logic}";
    $this->binds[$placeHolder] = $value;
    return $this;
}
```

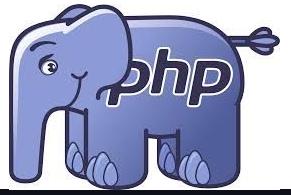
Método executeQuery



where: Adiciona uma condição à cláusula WHERE da query.

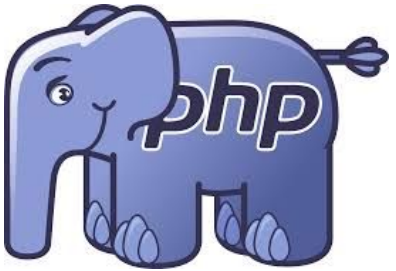
- **$field:** O campo que será comparado na condição.
- **$operator:** O operador de comparação (e.g., `=`, `>`, `<`).
- **$value:** O valor que será comparado ao campo.
- **$logic:** Operador lógico opcional (e.g., `AND`, `OR`) para encadear múltiplas condições.
- **$placeHolder:** Criado para vincular o valor na query. Se o campo contiver um ponto (`.`), apenas o nome após o ponto é usado como placeholder.
- **$this->where[]:** A condição é adicionada ao array `where`.
- **$this->binds[$placeHolder]:** O valor é armazenado no array `binds` com o placeholder como chave.
- **return $this:** Retorna a instância da classe para permitir encadeamento de métodos.

Método createQuery




```php
private function createQuery()
{
    if (!$this->table) {
        throw new \Exception("A consulta precisa invocar o método table.");
    }
    if (!$this->fieldsAndValues) {
        throw new \Exception("A consulta precisa dos dados para realizar a atualização.");
    }
    $query = '';
    $query = "update {$this->table} set ";
    foreach ($this->fieldsAndValues as $field => $value) {
        $query .= "{$field} = :{$field},";
        $this->binds[$field] = $value;
    }
    $query = rtrim($query, ',');
    $query .= (isset($this->where) and (count($this->where) > 0)) ? ' where ' . implode(' ', $this->where) : '';
    return $query;
}
```

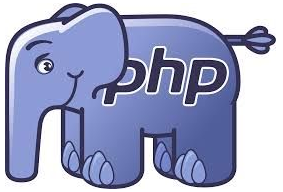
Método `createQuery`



createQuery: Constrói a query SQL de atualização.

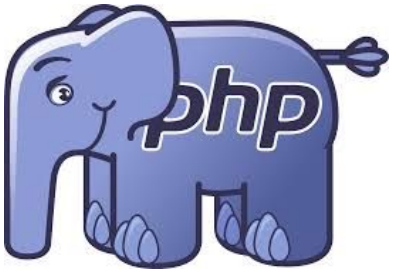
- **Verificação da tabela:** Lança uma exceção se a tabela não for definida.
- **Verificação dos campos:** Lança uma exceção se não houver campos e valores definidos para atualização.
- **Construção da query:** Monta a string SQL, adicionando o nome da tabela e os pares campo/valor.
- **rtrim:** Remove a última vírgula da query.
- **Cláusula WHERE:** Se houver condições definidas, elas são adicionadas à query.
- **return $query:** Retorna a query completa.

Método update




```php
public function update()
{
    $query = $this->createQuery();
    try {
        return $this->executeQuery($query);
    } catch (\PDOException $e) {
        throw new \Exception("Restrição: {$e->getMessage()}");
    }
}
```

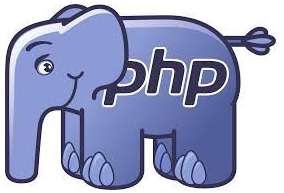
Método `update`



update: Método público que executa o processo de atualização.

- **$query:** A query é criada utilizando `createQuery()`.
- **try/catch:** A query é executada e, se houver um erro de PDO (banco de dados), uma exceção com uma mensagem específica é lançada.

Método `table`




```php
public static function table(string $table)
{
    $self = new self;
    $self->table = $table;
    return $self;
}
```

Método `table`

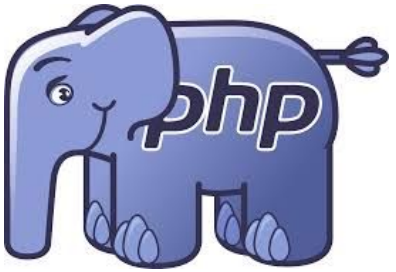
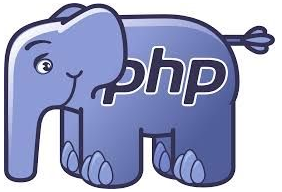


table: Define a tabela que será atualizada.

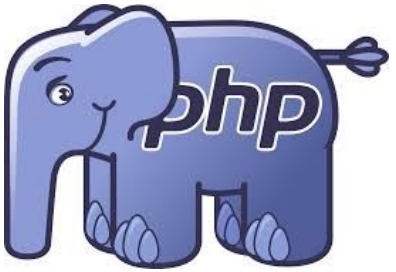
- **self:** Cria uma nova instância de `UpdateQuery`.
- **$self->table = $table:** Define o nome da tabela.
- **return $self:** Retorna a instância para permitir o encadeamento de métodos.

Método `set`




```php
public function set(array $fieldsAndValues)
{
    $this->fieldsAndValues = $fieldsAndValues;
    return $this;
}
```

Método `set`



set: Define os campos e valores que serão atualizados.

- **$fieldsAndValues:** Array associativo onde as chaves são os nomes dos campos e os valores são os novos valores.
- **$this->fieldsAndValues:** Atribui o array ao atributo da classe.
- **return $this:** Retorna a instância para permitir encadeamento de métodos.