DESEMPENHO DE APLICAÇÕES WEB

Prof. Me. Hélio Esperidião

Contexto

- Aplicações web podem ser acessadas no mundo inteiro por meio da internet.
- Grandes aplicações contam com milhares de acessos simultâneos.
- Quanto mais acessos... mais recursos de servidor são necessários
  - + ram
  - + processador
  - +banda

```php
public function readAll(){
    // Define a consulta SQL para selecionar todos os cargos ordenados por nome
    $SQL = "Select * from cargo order by nomeCargo";
    // Prepara a consulta
    $prepareSQL = Banco::getConexao()->prepare($SQL);
    // Executa a consulta
    $executou = $prepareSQL->execute();
    // Obtém o resultado da consulta
    $matrizTuplas = $prepareSQL->get_result();
    // Inicializa um vetor para armazenar os cargos
    $vetorCargos = array();
    $i = 0;
    // Itera sobre as tuplas do resultado
    while ($tupla = $matrizTuplas->fetch_object()) {
        // Cria uma nova instância de Cargo para cada tupla encontrada
        $vetorCargos[$i] = new Cargo();
        // Define o ID e o nome do cargo na instância
        $vetorCargos[$i]->setIdCargo($tupla->idCargo);
        $vetorCargos[$i]->setNomeCargo($tupla->nomeCargo);
        $i++;
    }
    // Retorna o vetor com os cargos encontrados
    return $vetorCargos;
}
```

Qual o problema de desempenho?

ps. Não significa que o código está errado

O problema:

- É gerada uma matriz de objetos de cargo.
- Essa matriz ocupa espaço de memória RAM(Memória primária).
  - Quanto mais cargos, mais memória é gasta.
- Imagine o mesmo código sendo adaptado para a carteira de clientes de um mega varejista brasileiro.
  - Quantos milhões de clientes?
  - Quanto de RAM será alocada?

```php
public function readAll(){
        $SQL = "Select * from cargo order by nomeCargo";
        $prepareSQL = Banco::getConexao()->prepare($SQL);    // Prepara a consulta
        $prepareSQL->execute();  // Executa a consulta
        $matrizTuplas = $prepareSQL->get_result(); // Obtém o resultado da consulta
        $vetorCargos = array();// Inicializa um vetor para armazenar os cargos
        $i = 0;
        while ($tupla = $matrizTuplas->fetch_object()) {
            if($tupla->idCargo==1 || $tupla->idCargo== 2) {
                $vetorCargos[$i] = new Cargo();
                $vetorCargos[$i]->setIdCargo($tupla->idCargo);
                $vetorCargos[$i]->setNomeCargo($tupla->nomeCargo);
                $i++;
            }
        }
        // Retorna o vetor com os cargos encontrados
        return $vetorCargos;
    }
```

Qual o problema de desempenho?

ps. O código roda e funciona, mas há um erro teórico crítico.

O problema:

- São buscados todos os cargos e é realizado um filtro dentro da estrutura de repetição.
- Se houver 1 milhão de cargos, a estrutura de repetição irá repetir por 1 milhão de vezes.

O que está melhor?
Quais as vantagens e desvantagens?

```php
public function readAll() {
        // Define a consulta SQL para selecionar todos os cargos ordenados por nome
        $SQL = "Select * from cargo order by nomeCargo";
        // Prepara a consulta
        $prepareSQL = Banco::getConexao()->prepare($SQL);
        // Executa a consulta
        $prepareSQL->execute();
        // Obtém o resultado da consulta
        $matrizTuplas = $prepareSQL->get_result();
       //MYSQLI_ASSOC indica que a matriz retornada será uma matriz associativa,
       //onde os nomes das colunas do banco de dados serão utilizados como chaves.
        $dados = $matrizTuplas->fetch_all(MYSQLI_ASSOC);//

        return $dados;
    }
```

Vantagens e desvantagens

- Vantagem
  - Consome muito menos memória.
  - Ideal para recuperar dados do banco e apresentar ao usuário sem que sejam realizadas operações adicionais.
- Desvantagem
  - Se operações adicionais são necessárias por meio da classe cargo, não será possível pois não há objetos da classe cargo.

O que está melhor?

```php
public function readAll() {
        // Define a consulta SQL para selecionar todos os cargos ordenados por nome
        $SQL = "Select * from cargo order by nomeCargo Where idCargo=1 or idCargo = 2";
        // Prepara a consulta
        $prepareSQL = Banco::getConexao()->prepare($SQL);
        $prepareSQL->execute();          // Executa a consulta
        // Obtém o resultado da consulta
        $matrizTuplas = $prepareSQL->get_result();
       //MYSQLI_ASSOC indica que a matriz retornada será uma matriz associativa,
       //onde os nomes das colunas do banco de dados serão utilizados como chaves.
        $dados = $matrizTuplas->fetch_all(MYSQLI_ASSOC);//

        return $dados;
    }
```

Desempenho do banco

- O banco de SGBD é preparado para realizar filtros complexos em velocidades extremamente rápidas.
- O banco SGBD é otimizado e utiliza técnicas de estruturas de dados avançadas para realizar consultas muito rapidamente(mais rápido do que passar por todos os dados usando uma estrutura de repetição simples).
  - Portanto, nunca faça filtros fora do SQL
  - Ps. NUNCAAAAAAAAAAAAAAAAAAAA
- Ps. **Curiosidade**: *Procure por arvores e estruturas de dados que são utilizadas para armazenar e filtrar dados*.

Agora tudo certo?

```
public function readAll() {
        // Define a consulta SQL para selecionar todos os cargos ordenados por nome
        $SQL = "Select * from cargo order by nomeCargo";
        // Prepara a consulta
        $prepareSQL = Banco::getConexao()->prepare($SQL);
        // Executa a consulta
        $prepareSQL->execute();
        // Obtém o resultado da consulta
        $matrizTuplas = $prepareSQL->get_result();
       //MYSQLI_ASSOC indica que a matriz retornada será uma matriz associativa,
       //onde os nomes das colunas do banco de dados serão utilizados como chaves.
        $dados = $matrizTuplas->fetch_all(MYSQLI_ASSOC);//

        return $dados;
    }
```

E se houver 1 milhão de cargos?

- Você irá exibir um milhão de cargos para o usuário?
- É necessário acessar todos esses cargos de uma única vez?
- E se eu acessar 1milhão de cargos de uma única vez?
  - `Select * from cargo order by nomeCargo`
  - O que acontece?

Nota de SQL

```
SELECT column1, column2, ...
FROM table_name
LIMIT offset, count;
```

offset: Especifica o número de registros a serem ignorados no início. É opcional e o padrão é 0.

count: Especifica o número máximo de registros a serem retornados. Se omitido, todos os registros a partir do deslocamento serão retornados.

Exemplo

- Suponha que temos uma tabela chamada cargo com os campos idCargo e nomeCargo, e queremos selecionar os 5 primeiros registros ordenados pelo nome do cargo:

- SELECT * FROM cargo ORDER BY nomeCargo LIMIT 0, 5;

Paginação

- A paginação em programação para web é um método usado para dividir grandes conjuntos de dados em páginas menores, permitindo que os usuários naveguem pelos dados de forma mais eficiente.
- Geralmente, é implementada em aplicativos da web que exibem listas de itens, como resultados de pesquisa, feeds de notícias, galerias de imagens, etc.

- Quando os conjuntos de dados são grandes demais para serem exibidos em uma única página, a paginação divide esses dados em páginas menores e exibe apenas uma página de cada vez.
  - Os usuários podem então navegar entre as páginas para visualizar diferentes conjuntos de dados.

Implementação de Paginação com LIMIT:

- Para implementar a paginação em uma consulta SQL, usamos a cláusula LIMIT com valores dinâmicos para o offset e count.
- Por exemplo, para exibir a página 1 com 10 registros por página:

- SELECT * FROM cargo ORDER BY nomeCargo LIMIT 0, 10;
  - 0: são ignorados zero tuplas
  - 10: são selecionadas 10 tuplas.

Implementação de Paginação com LIMIT: página 2

- Para implementar a paginação em uma consulta SQL, usamos a cláusula LIMIT com valores dinâmicos para o offset e count.
- Por exemplo, para exibir a página 2 com 10 registros por página:
  - SELECT * FROM cargo ORDER BY nomeCargo LIMIT 10, 10;
- Isso pulará os primeiros 10 registros e retornará os próximos 10 registros, que são os registros da página 2.

E a terceira página

- Supondo que já sabemos que cada página exibe 10 registros, precisamos calcular o deslocamento (offset) correto para a página 3.
- O deslocamento é calculado multiplicando o número da página pelo número de registros por página e subtraindo o número de registros por página.
- Então, para a página 3:
  - Deslocamento (offset) = (3 - 1) * 10 = 20
- Agora, podemos usar a cláusula LIMIT para selecionar os registros da página 3:
  - SELECT * FROM cargo ORDER BY nomeCargo LIMIT 20, 10;

Vamos implementar na API

Observe a uri:

- /cargos/paginas/1
  - Primeira página de cargos.
- /cargos/paginas/2
  - Segunda página de cargos.
- /cargos/paginas/3
  - Terceira página de cargos.
- É intuitivo e ainda é preservada a arquitetura REST API,
- Ps. *Os escritórios de programação e desenvolvimento tendem a adotar suas próprias regras, mas a solução apresentada é simples, elegante e atende as características de uma arquitetura moderna REST API.*

Rota:

```
// Define uma rota para recuperar uma página de cargos
$roteador->get("/cargos/paginas/(\d+)/", function ($pagina) {
    require_once ("controle/cargo/controle_cargo_read_by_page.php");
});
```

/cargos/paginas/2

```php
public function readByPage($pagina){
    // Definir o número de itens por página
    $itensPorPagina = 5;
    // Calcular o início dos registros com base na página atual
    $inicio = ($pagina - 1) * $itensPorPagina;

    // Define a consulta SQL para selecionar os registros da página atual
    $SQL = "SELECT * FROM cargo ORDER BY nomeCargo LIMIT ?, ?";
    // Prepara a consulta
    $prepareSQL = Banco::getConexao()->prepare($SQL);
    // Vincula os parâmetros da consulta (início e número de itens por página)
    $prepareSQL->bind_param('ii', $inicio, $itensPorPagina);
    // Executa a consulta
    $executou = $prepareSQL->execute();
    // Obtém o resultado da consulta
    $matrizTuplas = $prepareSQL->get_result();
    // Transforma o resultado em um array associativo
    $dados = $matrizTuplas->fetch_all(MYSQLI_ASSOC);
    return $dados;
}
```

Faça uma análise

controle_cargo_read_by_page.php

```php
require_once ("modelo/Cargo.php");
// Cria um novo objeto para armazenar a resposta
$objResposta = new stdClass();
// Cria um novo objeto da classe Cargo
$objetoCargo = new Cargo();
// Obtém todos os cargos do banco de dados
$vetor = $objetoCargo ->readByPage($pagina);
// Define o código de resposta como 1
$objResposta->cod = 1;
// Define o status da resposta como verdadeiro
$objResposta->status = true;
// Define a mensagem de sucesso
$objResposta->msg = "executado com sucesso";
// Define o vetor de cargos na resposta
$objResposta->cargos = $vetor;

// Define o código de status da resposta como 200 (OK)
header("HTTP/1.1 200");
// Define o tipo de conteúdo da resposta como JSON
header("Content-Type: application/json");
// Converte o objeto resposta em JSON e o imprime na saída
echo json_encode($objResposta);
```