1. Princípios Gerais para Código Rápido

1.1 Escolha Algoritmos e Estruturas de Dados Adequados

- **Regra:** Sempre escolha algoritmos e estruturas de dados apropriados ao problema.
- **Como aplicar:**
    - Use listas encadeadas para inserções frequentes no meio de coleções.
    - Prefira tabelas hash (como arrays associativos no PHP) para buscas rápidas.
    - Evite algoritmos com alta complexidade de tempo como $O(n2)O(n^2)$O(n2), substituindo por alternativas $O(nlog⁡n)O(n \log n)$O(nlogn) ou $O(n)O(n)$O(n) sempre que possível.

1.2 Reduza Operações Repetitivas

- **Regra:** Evite cálculos e operações desnecessárias em loops ou funções.
- **Como aplicar:**
    - **Antes:**

```
php
Copiar código
for ($i = 0; $i < count($array); $i++) {
    // lógica
}
```

    - **Depois:**

```
php
Copiar código
$length = count($array);
for ($i = 0; $i < $length; $i++) {
    // lógica
}
```

1.3 Minimize I/O e Consultas ao Banco de Dados

- **Regra:** Operações de entrada/saída (I/O) e consultas são caras; reduza seu uso ao mínimo.
- **Como aplicar:**

- Prefira carregar dados necessários em um lote único (`JOIN`, consultas otimizadas).
- Use cache para resultados de consultas frequentes.

1.4 Evite Alocações Excessivas de Memória

- **Regra:** Use estruturas e métodos que minimizem cópias desnecessárias de dados.
- **Como aplicar:**
  - Utilize referências (&) ao invés de criar cópias em PHP quando manipular arrays ou objetos grandes.

1.5 Otimize Estruturas Condicionais

- **Regra:** Organize condições por frequência e probabilidade.
- **Como aplicar:**
  - **Antes:**

```
php
Copiar código
if ($x === 'rare_case') { ... }
elseif ($x === 'common_case') { ... }
```

  - **Depois:**

```
php
Copiar código
if ($x === 'common_case') { ... }
elseif ($x === 'rare_case') { ... }
```

2. Técnicas de Otimização

2.1 Cache

- **Quando aplicar:** Sempre que operações repetidas puderem ser armazenadas temporariamente.
- **Como aplicar:**
  - Use mecanismos como Redis ou Memcached para cache de consultas.
  - Armazene resultados em variáveis locais para evitar recomputação.

2.2 Paginação

- **Quando aplicar:** Ao trabalhar com grandes conjuntos de dados.
- **Como aplicar:**
    - Carregue apenas os registros necessários para cada página.
    - Exemplo com SQL:

```
sql
Copiar código
SELECT * FROM documents LIMIT 50 OFFSET 0;
```

2.3 Lazy Loading

- **Quando aplicar:** Para carregar apenas os dados necessários, economizando memória.
- **Como aplicar:**
    - Em frameworks ORM (como Doctrine), ative o carregamento preguiçoso em relações.
    - Carregue arquivos ou classes apenas quando forem usados (autoloader no PHP).

2.4 Paralelismo e Assincronismo

- **Quando aplicar:** Em tarefas independentes que podem ser realizadas em paralelo.
- **Como aplicar:**
    - Utilize filas de processamento como RabbitMQ para delegar tarefas.
    - Use threads em sistemas compatíveis ou paralelismo nativo do PHP com `pthreads`.

2.5 Compressão

- **Quando aplicar:** Para reduzir tamanho de dados transferidos em APIs ou armazenamento.
- **Como aplicar:**
    - Comprime dados JSON ou XML.
    - Use algoritmos como Gzip para comprimir respostas HTTP.

3. Ferramentas para Medir Desempenho

3.1 Profiler

- Use ferramentas como **Xdebug** ou **Blackfire** para identificar gargalos no código.

3.2 Logs e Monitoramento

- Implemente logs de tempo em partes críticas do código para identificar lentidão.
- Use ferramentas como New Relic ou Grafana.

4. Exemplos Práticos

4.1 Otimização de Consultas ao Banco

Antes:

Consulta dentro de um loop:

```
php
Copiar código
foreach ($users as $user) {
    $orders = $db->query("SELECT * FROM orders WHERE user_id =
{$user['id']}");
}
```

Depois:

Use uma única consulta com `IN`:

```
php
Copiar código
$userIds = array_column($users, 'id');
$orders = $db->query("SELECT * FROM orders WHERE user_id IN (" .
implode(',', $userIds) . ")");
```

4.2 Uso de Cache

Antes:

Recomputando o mesmo valor:

```
php
Copiar código
for ($i = 0; $i < 1000; $i++) {
    $result = expensiveCalculation($i);
    echo $result;
}
```

Depois:

Armazene o resultado:

```
php
Copiar código
$cache = [];
for ($i = 0; $i < 1000; $i++) {
    if (!isset($cache[$i])) {
        $cache[$i] = expensiveCalculation($i);
    }
    echo $cache[$i];
}
```

4.3 Processamento Assíncrono

Antes:

Processo bloqueante:

```
php
Copiar código
$report = generateReport();
sendEmail($report);
```

Depois:

Desacople o envio de relatórios usando uma fila:

```
php
Copiar código
$report = generateReport();
dispatchToQueue('sendEmail', $report);
```

5. Aplicação em Cenários Reais

5.1 APIs

- Retorne apenas os dados necessários (use `fields` para seleção).
- Compacte respostas com Gzip.

5.2 Interfaces de Usuário

- Use *defer* e *async* em scripts JS para melhorar carregamento.
- Implemente carregamento infinito (infinite scroll) para grandes listas.

5.3 Arquitetura

- Prefira microsserviços para segmentar responsabilidades.
- Distribua cargas pesadas com balanceamento de carga e servidores de cache.

6. Práticas Avançadas

6.1 Parallel Streams

Utilize múltiplas threads ou streams:

```
php
Copiar código
use parallel\Runtime;

$runtime = new Runtime();
$future = $runtime->run(function() {
    return expensiveTask();
});
```

```
$result = $future->value();
```

6.2 Preloading

Pré-carregue scripts ou classes frequentemente usados com `opcache` em PHP.