Relacionamentos de Banco de Dados com Laravel Eloquent

Um guia passo a passo para implementar relacionamentos 1:1, 1:N e N:N com SQL puro

Estrutura das Tabelas

Tabela Alunos

Campo	Tipo
id	INT AUTO_INCREMENT
nome	VARCHAR(100)
email	VARCHAR(100) UNIQUE
idade	INT

Tabela Cursos

Campo	Tipo
id	INT AUTO_INCREMENT
nome	VARCHAR(100)

Tabela Turmas

Campo	Tipo
id	INT AUTO_INCREMENT
descricao	VARCHAR(100)
curso_id	INT (chave estrangeira)

Tabela Turma_Aluno

Campo	Tipo
id	INT AUTO_INCREMENT
aluno_id	INT (chave estrangeira)
turma_id	INT (chave estrangeira)

Estas tabelas serão utilizadas para demonstrar os diferentes tipos de relacionamentos no Laravel Eloquent.

SQL para Criação das Tabelas

Tabela Alunos

SQL - Alunos

```sql
CREATE TABLE alunos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    idade INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Tabela Turmas

SQL - Turmas

```sql
CREATE TABLE turmas (
    id INT AUTO_INCREMENT PRIMARY KEY,
    descricao VARCHAR(100) NOT NULL,
    curso_id INT NOT NULL,
    FOREIGN KEY (curso_id)
        REFERENCES cursos(id)
);
```

Tabela Cursos

SQL - Cursos

```sql
CREATE TABLE cursos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Tabela Turma_Aluno

SQL - Turma_Aluno

```sql
CREATE TABLE turma_aluno (
    id INT AUTO_INCREMENT PRIMARY KEY,
    aluno_id INT NOT NULL,
    turma_id INT NOT NULL,
    FOREIGN KEY (aluno_id)
        REFERENCES alunos(id),
    FOREIGN KEY (turma_id)
        REFERENCES turmas(id)
);
```

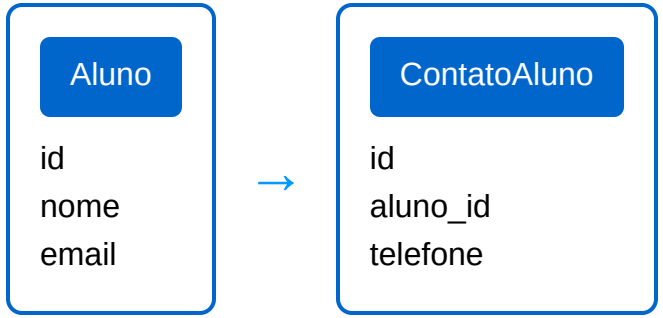
Observe o uso de **FOREIGN KEY** para estabelecer os relacionamentos entre as tabelas, sem a cláusula CASCADE.

Relacionamento Um para Um (1:1)

No relacionamento **Um para Um (1:1)**, cada registro em uma tabela está associado a exatamente um registro em outra tabela, e vice-versa.

Cenário de Exemplo

Um aluno tem um e apenas um perfil de contato, e um perfil de contato pertence a apenas um aluno.



SQL para Tabela de Contatos

```sql
CREATE TABLE contato_alunos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    aluno_id INT NOT NULL UNIQUE,
    telefone VARCHAR(20),
    endereco VARCHAR(200),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (aluno_id)
        REFERENCES alunos(id)
);
```

Pontos importantes:

- A restrição **UNIQUE** na chave estrangeira garante o relacionamento 1:1
- A chave estrangeira fica na tabela que "pertence" à outra (ContatoAluno)
- Sem a cláusula CASCADE, você precisa gerenciar manualmente a exclusão de registros relacionados

Implementação do Modelo 1:1

Modelo Aluno.php

Aluno.php

```
namespace App\Models;

class Aluno extends Model
{
    protected $fillable = ['nome', 'email', 'idade'];

    public function contatoAluno()
    {
        return $this->hasOne(ContatoAluno::class);
    }
}
```

O método **hasOne()** define que este modelo possui um relacionamento com outro modelo.

Modelo ContatoAluno.php

ContatoAluno.php

```
namespace App\Models;

class ContatoAluno extends Model
{
    protected $fillable = ['aluno_id', 'telefone', 'endereco'];

    public function aluno()
    {
        return $this->belongsTo(Aluno::class);
    }
}
```

O método **belongsTo()** define que este modelo pertence a outro modelo.

Importante:

- O modelo que possui a chave estrangeira usa **belongsTo()**
- O modelo referenciado pela chave estrangeira usa **hasOne()**

Uso do Relacionamento 1:1

Criar Aluno com Contato

```
// Criar aluno
$aluno = Aluno::create([
    'nome' => 'João Silva',
    'email' => 'joao@example.com',
    'idade' => 20
]);

// Criar contato para o aluno
$aluno->contatoAluno()->create([
    'telefone' => '11987654321',
    'endereco' => 'Rua A, 123'
]);
```

Acessar Contato do Aluno

```
// Obter o contato do aluno
$contato = $aluno->contatoAluno;

// Acessar propriedades
echo $contato->telefone;
```

Acessar Aluno a partir do Contato

```
// Encontrar um contato
$contato = ContatoAluno::find(1);

// Acessar o aluno relacionado
$aluno = $contato->aluno;

// Acessar propriedades
echo $aluno->nome;
```

Verificar e Atualizar

```
// Verificar se existe
if ($aluno->contatoAluno) {
    // Atualizar contato existente
    $aluno->contatoAluno->update([
        'telefone' => '11999998888'
    ]);
}
```

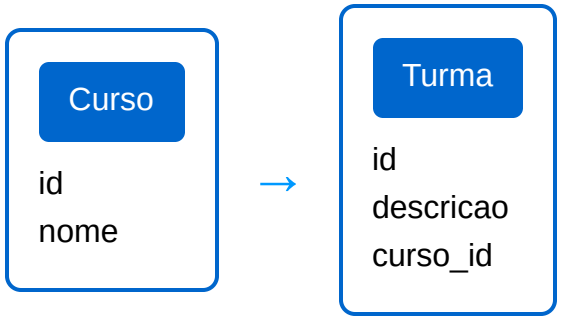
O relacionamento **Um para Um** é ideal para informações complementares que pertencem logicamente a entidades separadas.

Relacionamento Um para Muitos (1:N)

No relacionamento **Um para Muitos (1:N)**, um registro em uma tabela pode estar associado a vários registros em outra tabela, mas cada registro na segunda tabela está associado a apenas um registro na primeira.

Cenário de Exemplo

Um curso pode ter várias turmas, mas cada turma pertence a apenas um curso.



SQL para Tabela de Turmas

```
CREATE TABLE turmas (
    id INT AUTO_INCREMENT PRIMARY KEY,
    descricao VARCHAR(100) NOT NULL,
    curso_id INT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (curso_id)
        REFERENCES cursos(id)
);
```

Características importantes:

- A chave estrangeira **curso_id** estabelece a relação entre a turma e seu curso
- Sem a cláusula CASCADE, você precisa gerenciar manualmente a exclusão de turmas quando um curso for excluído
- Não há restrição de unicidade na chave estrangeira, permitindo múltiplas turmas para um mesmo curso

Implementação do Modelo 1:N

Modelo Curso.php

Curso.php

```php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Curso extends Model
{
    protected $fillable = ['nome'];

    public function turmas()
    {
        return $this->hasMany(Turma::class);
    }
}
```

O método **hasMany()** define que:

- Um curso pode ter várias turmas
- A chave estrangeira está na tabela relacionada
- Por convenção, procura por **curso_id** na tabela turmas

Modelo Turma.php

Turma.php

```php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Turma extends Model
{
    protected $fillable = [
        'descricao',
        'curso_id'
    ];

    public function curso()
    {
        return $this->belongsTo(Curso::class);
    }
}
```

O método **belongsTo()** define que:

- Uma turma pertence a um único curso
- A chave estrangeira está nesta tabela
- Por convenção, procura por **curso_id** nesta tabela

Os métodos **hasMany()** e **belongsTo()** são complementares e formam as duas extremidades do relacionamento 1:N.

Uso do Relacionamento 1:N

Criar Curso com Turmas

```
// Criar um curso
$curso = Curso::create([
    'nome' => 'Engenharia de Software'
]);

// Adicionar turmas ao curso
$curso->turmas()->create([
    'descricao' => 'Turma A'
]);
$curso->turmas()->create([
    'descricao' => 'Turma B'
]);
```

Acessar Turmas de um Curso

```
// Obter todas as turmas do curso
$turmas = $curso->turmas;

// Iterar sobre as turmas
foreach ($turmas as $turma) {
    echo $turma->descricao;
}
```

Acessar Curso de uma Turma

```
// Encontrar uma turma
$turma = Turma::find(1);

// Acessar o curso relacionado
$curso = $turma->curso;

// Acessar propriedades do curso
echo $curso->nome;
```

Consultas Eficientes

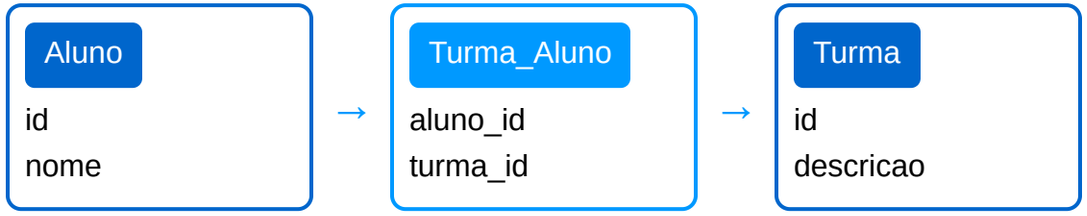
```
// Encontrar cursos com turmas
$cursosComTurmas = Curso::has('turmas')->get();

// Carregamento antecipado (eager loading)
$cursos = Curso::with('turmas')->get();
```

O relacionamento **Um para Muitos** é ideal para representar hierarquias e agrupamentos de dados relacionados.

Relacionamento Muitos para Muitos (N:N)

No relacionamento **Muitos para Muitos (N:N)**, registros em ambas as tabelas podem estar associados a vários registros na outra tabela.



SQL para Tabela Pivô

```sql
CREATE TABLE turma_aluno (
    id INT AUTO_INCREMENT PRIMARY KEY,
    aluno_id INT NOT NULL,
    turma_id INT NOT NULL,
    FOREIGN KEY (aluno_id) REFERENCES alunos(id),
    FOREIGN KEY (turma_id) REFERENCES turmas(id)
);
```

Definição dos Modelos

```php
// Aluno.php
public function turmas()
{
    return $this->belongsToMany(
        Turma::class,      // Modelo relacionado
        'turma_aluno',     // Tabela pivô
        'aluno_id',        // Chave estrangeira do modelo atual
        'turma_id'         // Chave estrangeira do modelo relaci
    );
}
```

```php
// Turma.php
public function alunos()
{
    return $this->belongsToMany(
        Aluno::class,      // Modelo relacionado
        'turma_aluno',     // Tabela pivô
        'turma_id',        // Chave estrangeira do modelo atual
        'aluno_id'         // Chave estrangeira do modelo relaci
    );
}
```

O método **belongsToMany()** permite especificar explicitamente o nome da tabela pivô e as chaves estrangeiras.

Exibindo o Curso da Turma

Template Blade para Detalhes da Turma

resources/views/turmas/show.blade.php

```
<div class="card">
    <div class="card-header">
        <h2>Detalhes da Turma: {{ $turma->descricao }}</h2>
    </div>
    <div class="card-body">
        <h3>Curso</h3>
        <p>
            <strong>Nome do Curso:</strong>
            {{ $turma->curso->nome }}
        </p>

        <h3>Alunos Matriculados</h3>
        <ul>
            @forelse ($turma->alunos as $aluno)
                <li>{{ $aluno->nome }}
                    ({{ $aluno->email }})
                </li>
            @empty
                <li>Nenhum aluno matriculado.</li>
            @endforelse
        </ul>
    </div>
</div>
```

Observe como acessamos o curso relacionado através de **$turma->curso->nome**, graças ao relacionamento definido no modelo.

Controller para Exibir a Turma

TurmaController.php

```
namespace App\Http\Controllers;

use App\Models\Turma;
use Illuminate\Http\Request;

class TurmaController extends Controller
{
    public function show($id)
    {
        // Carrega a turma com seu curso e alunos
        $turma = Turma::with(['curso', 'alunos'])
                      ->findOrFail($id);

        return view('turmas.show',
                    compact('turma'));
    }
}
```

Usando **with(['curso', 'alunos'])** carregamos antecipadamente os relacionamentos para evitar o problema N+1.

Dicas para templates:

- Use **@forelse/@empty** para lidar com coleções vazias
- Sempre carregue relacionamentos com **with()** no controller
- Verifique se o relacionamento existe antes de acessá-lo: **@if($turma->curso)**

Exibindo as Turmas do Aluno

Template Blade para Perfil do Aluno

```
<div class="card">
  <div class="card-header">
    <h2>Perfil do Aluno: {{ $aluno->nome }}</h2>
  </div>
  <div class="card-body">
    <p><strong>Email:</strong> {{ $aluno->email }}</p>
    <p><strong>Idade:</strong> {{ $aluno->idade }} anos</p>

    @if ($aluno->contatoAluno)
      <h3>Informações de Contato</h3>
      <p><strong>Telefone:</strong> {{ $aluno->contatoAluno-
      <p><strong>Endereço:</strong> {{ $aluno->contatoAluno-
    @endif

    <h3>Turmas Matriculadas</h3>
    <ul>
      @forelse ($aluno->turmas as $turma)
        <li>
          {{ $turma->descricao }}
          <small>(Curso: {{ $turma->curso->nome }})</small>
        </li>
      @empty
        <li>Aluno não está matriculado em nenhuma turma.</li
      @endforelse
    </ul>
  </div>
</div>
```

Observe como acessamos o curso de cada turma usando **$turma->curso->nome**, navegando através dos relacionamentos.

Controller para Exibir Aluno com Turmas

```
namespace App\Http\Controllers;

use App\Models\Aluno;
use Illuminate\Http\Request;

class AlunoController extends Controller
{
    public function show($id)
    {
        // Carrega o aluno com suas turmas, os cursos
        // das turmas e seu contato
        $aluno = Aluno::with([
            'turmas.curso',
            'contatoAluno'
        ])->findOrFail($id);


        return view('alunos.show', compact('aluno'));
    }
}
```

Carregamento Aninhado de Relacionamentos

```
// Carregamento aninhado com dot notation
$aluno = Aluno::with('turmas.curso')->find($id);

// Acessando dados aninhados
foreach ($aluno->turmas as $turma) {
    echo "Turma: " . $turma->descricao;
    echo "Curso: " . $turma->curso->nome;
}
```

O método **with()** permite carregar relacionamentos aninhados usando a notação de ponto, evitando o problema N+1 de consultas.