Laravel RBAC — Using `role_id` Directly on the `users` Table

1) Goal

This document explains an RBAC approach where each user has **exactly one role**, by storing `role_id` directly on the `users` table.

This is the simplest possible RBAC structure and works well for many small-to-medium Laravel apps.

---

2) When This Approach is Best

Use `users.role_id` if: - Each user can only have **one** role - Your roles are global (not per org/project) - You want minimal complexity - Your system is something like: - Admin - Editor - Viewer

Examples: - Internal company dashboard - Simple CMS - Small e-commerce admin

---

3) When NOT to Use This Approach

Avoid `users.role_id` if: - Users need **multiple roles** - You need roles **per tenant/org/project** - You want future-proof RBAC

Example problem: - A user is Viewer globally, but Admin in Org #5

This cannot be represented with a single `role_id` column.

---

4) Database Design

Tables

`roles`

Stores role definitions.

`privileges`

Stores permission definitions.

`role_privilege`

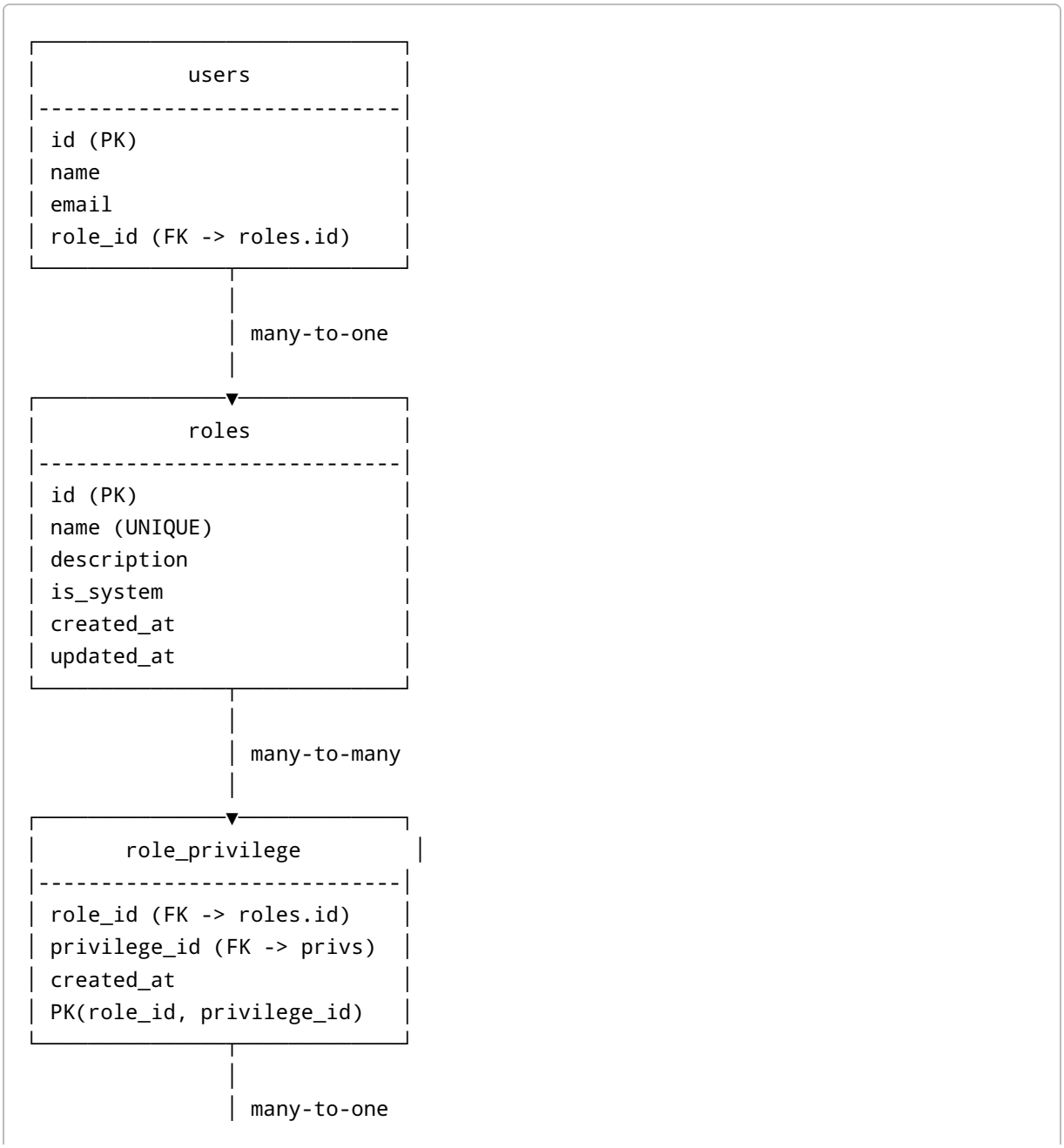
Pivot that connects roles to privileges.

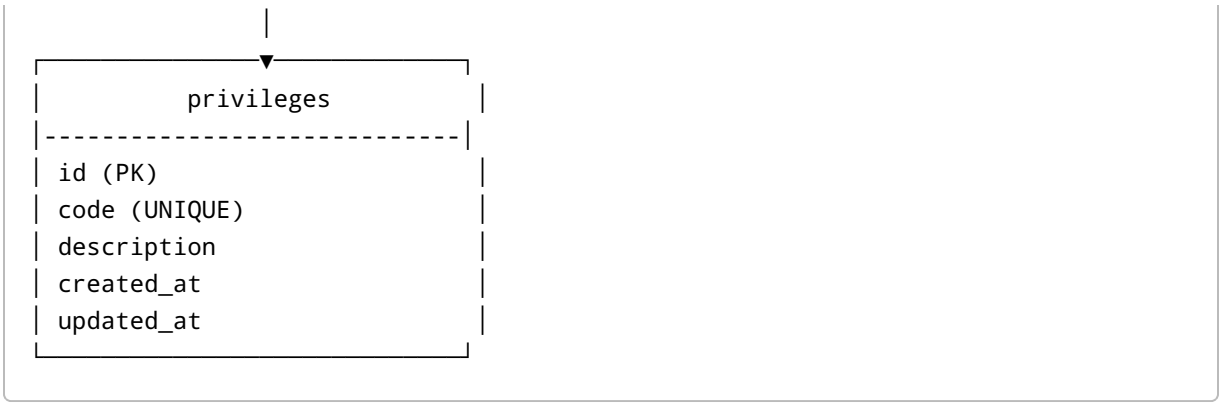
`users`

Contains a direct `role_id` column.

---

5) ERD Chart





---

6) Laravel Migrations

6.1 `create_roles_table.php`

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up(): void
    {
        Schema::create('roles', function (Blueprint $table) {
            $table->id();
            $table->string('name', 100)->unique();
            $table->text('description')->nullable();
            $table->boolean('is_system')->default(false);
            $table->timestamps();
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('roles');
    }
};
```

6.2 `create_privileges_table.php`

```php
<?php
```

```php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up(): void
    {
        Schema::create('privileges', function (Blueprint $table) {
            $table->id();
            $table->string('code', 150)->unique(); // e.g. users.read
            $table->text('description')->nullable();
            $table->timestamps();
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('privileges');
    }
};
```

6.3 `create_role_privilege_table.php`

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up(): void
    {
        Schema::create('role_privilege', function (Blueprint $table) {
            $table->foreignId('role_id')->constrained('roles')-
>cascadeOnDelete();
            $table->foreignId('privilege_id')->constrained('privileges')-
>cascadeOnDelete();
            $table->timestamp('created_at')->useCurrent();

            $table->primary(['role_id', 'privilege_id']);
            $table->index('privilege_id');
        });
    }

    public function down(): void
    {
```

```
        Schema::dropIfExists('role_privilege');
    }
};
```

6.4 Add `role_id` to users table

Create:

```
php artisan make:migration add_role_id_to_users_table --table=users
```

Migration file:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up(): void
    {
        Schema::table('users', function (Blueprint $table) {
            $table->foreignId('role_id')
                ->nullable()
                ->after('id')
                ->constrained('roles')
                ->nullOnDelete();

            $table->index('role_id');
        });
    }

    public function down(): void
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dropForeign(['role_id']);
            $table->dropIndex(['role_id']);
            $table->dropColumn('role_id');
        });
    }
};
```

---

7) Eloquent Relationships

In `User.php`

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

public function hasPrivilege(string $code): bool
{
    return $this->role()
        ->whereHas('privileges', fn ($q) => $q->where('code', $code))
        ->exists();
}
```

In `Role.php`

```
public function privileges()
{
    return $this->belongsToMany(Privilege::class, 'role_privilege');
}

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

---

8) Seeder Recommendation

Seed: - Roles: Admin, Editor, Viewer - Privileges: users.read, users.write, content.read, content.write

Admin role gets all privileges.

---

9) Summary

This design is ideal if:

- One user = one role
- Roles are global
- You want the simplest RBAC possible

If you later need multiple roles:

You will migrate from `users.role_id` to a pivot table `role_user`.

If you want, I can also provide: - a ready-to-run `RbacSeeder` - middleware (`privilege:users.write`) - policies/gates setup