Soft Deletes in Laravel: Restore, Force Delete, and Prune Data

Soft Deletes in Laravel: Restore, Force Delete, and Prune Data

Soft deletes let you “delete” rows without losing them immediately. Instead of removing data, Eloquent sets a `deleted_at` timestamp and excludes those rows from normal queries. You can later *restore* or *permanently remove* them, and even *prune* old soft-deleted data on a schedule. In this guide, you’ll enable soft deletes, build a Recycle Bin UI, and automate cleanup safely.

1 – Add `deleted_at` to Your Table

Add a soft delete column using the schema builder. Use `softDeletes()` (or `softDeletesTz()` if you prefer timezone-aware timestamps).

```
//
database/migrations/2025_08_27_000000_add_soft_deletes_to_posts_table.
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('posts', function (Blueprint $table) {
            $table->softDeletes(); // adds nullable deleted_at
TIMESTAMP
```

```
        });
    }

    public function down(): void
    {
        Schema::table('posts', function (Blueprint $table) {
            $table->dropSoftDeletes(); // drops deleted_at
        });
    }
};Code language: PHP (php)
```

This migration adds a `deleted_at` column that Eloquent uses to hide “trashed” rows. The `down()` method makes the change reversible.

Run the migration:

```
php artisan migrateCode language: Bash (bash)
```

After running, the table is ready to support soft deletes without breaking existing queries.

2 – Enable Soft Deletes on the Model

Add the `SoftDeletes` trait to your Eloquent model. This automatically excludes trashed rows from default queries such as `Model::all()`.

```
// app/Models/Post.php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Post extends Model
{
```

```
    use SoftDeletes;

    protected $fillable = ['user_id','title','body','status'];
}Code language: PHP (php)
```

With the trait, calling `Post::query()` ignores rows where `deleted_at` is not null. You'll use special helpers to include or filter trashed rows when needed.

3 – Soft Delete, Restore, and Force Delete

Soft delete marks the row; restore brings it back; force delete removes it permanently from the database.

```
// Soft delete a post
$post = Post::findOrFail($id);
$post->delete(); // sets deleted_at

// Restore a soft-deleted post
$post = Post::withTrashed()->findOrFail($id);
$post->restore(); // clears deleted_at

// Permanently delete
$post = Post::withTrashed()->findOrFail($id);
$post->forceDelete(); // removes row from DBCode language: PHP (php)
```

Use `withTrashed()` to access items regardless of deletion state, then call `restore()` or `forceDelete()` as appropriate.

4 – Query Helpers for Trashed Rows

These helpers give you fine control over which rows are returned in queries.

```
// Include trashed + non-trashed
$all = Post::withTrashed()->latest()->paginate(10);

// Only trashed
$trashed = Post::onlyTrashed()->orderBy('deleted_at','desc')->get();

// Explicitly exclude trashed (same as default)
$active = Post::withoutTrashed()->get();Code language: PHP (php)
```

`withTrashed()` is useful for admin reports; `onlyTrashed()` powers a Recycle Bin; `withoutTrashed()` matches the default behavior when the trait is enabled.

5 – Routes & Controller for a Recycle Bin UI

Expose routes to view trashed items, restore them, or permanently delete them. Authorize these actions to admin roles only.

```
// routes/web.php (snippet)
use App\Http\Controllers\PostTrashController;

Route::middleware(['auth'])->group(function () {
    Route::get('/posts/trash', [PostTrashController::class,
'index'])->name('posts.trash.index');
    Route::patch('/posts/{id}/restore', [PostTrashController::class,
'restore'])->name('posts.trash.restore');
    Route::delete('/posts/{id}/force', [PostTrashController::class,
'force'])->name('posts.trash.force');
});Code language: PHP (php)
```

These routes provide a list view plus endpoints to restore or force delete a specific post. Use policies/middleware to ensure only privileged users can perform destructive actions.

```
// app/Http/Controllers/PostTrashController.php
namespace App\Http\Controllers;

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

class PostTrashController extends Controller
{
    public function index()
    {
        $posts = Post::onlyTrashed()
            ->orderBy('deleted_at','desc')
            ->paginate(10);

        return view('posts.trash', compact('posts'));
    }

    public function restore($id)
    {
        $post = Post::withTrashed()->findOrFail($id);
        // $this->authorize('restore', $post); // optional policy
        $post->restore();

        return back()->with('status','Post restored.');
    }

    public function force($id)
    {
        $post = Post::withTrashed()->findOrFail($id);
        // $this->authorize('forceDelete', $post); // optional policy
        $post->forceDelete();

        return back()->with('status','Post permanently deleted.');
    }
}Code language: PHP (php)
```

The controller paginates trashed posts and provides RESTful handlers to restore or permanently delete items. Optionally enforce policies for extra safety.

6 – UI: Recycle Bin Blade View

Here’s a simple Recycle Bin with Restore and Delete buttons. Use CSRF and method spoofing to protect the actions.

```
<!-- resources/views/posts/trash.blade.php -->
@extends('layouts.app')

@section('content')
<div class="container">
  <h1 class="mb-4">Recycle Bin</h1>

  @if(session('status'))
    <div class="alert alert-success">{{ session('status') }}</div>
  @endif

  @forelse($posts as $post)
    <div class="card mb-3">
      <div class="card-body d-flex justify-content-between align-
items-center">
        <div>
          <h5 class="card-title mb-1">{{ $post->title }}</h5>
          <small class="text-muted">Deleted at: {{ $post->deleted_at
}}</small>
        </div>

        <div class="d-flex gap-2">
          <form method="POST" action="{{ route('posts.trash.restore',
$post->id) }}">
            @csrf @method('PATCH')
```

```
            <button class="btn btn-outline-secondary">Restore</button>
          </form>

          <form method="POST" action="{{ route('posts.trash.force',
$post->id) }}"
                onsubmit="return confirm('Permanently delete this
post?');">
            @csrf @method('DELETE')
            <button class="btn btn-danger">Delete Forever</button>
          </form>
        </div>
      </div>
    </div>
  @empty
    <p class="text-muted">No trashed posts.</p>
  @endforelse

  {{ $posts->links() }}
</div>
@endsectionCode language: PHP (php)
```

This UI lists trashed posts with the deletion timestamp and action buttons. Restore unsets `deleted_at`, while “Delete Forever” removes the row from the database.

7 – Scheduled Pruning of Old Soft-Deleted Rows

Use model pruning to automatically purge items that have been soft-deleted for a while (e.g., 30 days).

```
// app/Models/Post.php (add trait & prunable() if you like)
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
```

```
use Illuminate\Database\Eloquent\SoftDeletes;
use Illuminate\Database\Eloquent\Prunable;

class Post extends Model
{
    use SoftDeletes, Prunable;

    public function prunable()
    {
        // prune items soft-deleted more than 30 days ago
        return static::onlyTrashed()
            ->where('deleted_at', '<', now()->subDays(30));
    }
}Code language: PHP (php)
```

The `Prunable` trait defines a query selecting candidates for permanent removal. Here we keep trashed posts for 30 days before pruning them.

```
// app/Console/Kernel.php (schedule pruning)
protected function schedule(\Illuminate\Console\Scheduling\Schedule
$schedule): void
{
    // Run daily at 02:00; add --pretend in staging to preview
deletions
    $schedule->command('model:prune', [
        '--model' => [\App\Models\Post::class],
    ])->dailyAt('02:00');
}Code language: PHP (php)
```

This schedules the built-in `model:prune` command to run daily. In non-production environments, consider `--pretend` to preview what would be deleted.

Remember to set up your cron to run the Laravel scheduler: `* * * * * php /path/to/artisan schedule:run >> /dev/null 2>&1`.

Wrapping Up

Soft deletes provide a safe middle ground between “active” and “gone.” You added a `deleted_at` column, enabled the `SoftDeletes` trait, implemented restore and force delete flows, built a Recycle Bin UI, and configured pruning to keep the database lean. This pattern reduces accidental loss and gives you clean, auditable lifecycle management for records.

What’s Next

- Handling Large Data Sets with Chunking & Cursors
- How to Use Eloquent API Resources for Clean APIs
- Filtering and Searching with Eloquent Query Builder