Docs

ETL Project

Derafu ETL

Anonymous · 21/08/2026 · #php

Table of Contents

ETL Project

Derafu ETL

Anonymous · 21/08/2026 · #php

Derafu ETL

Last updated on 21/08/2026

Introduction

From Spreadsheets to Databases Seamlessly

Anonymous · 21/08/2026

From Spreadsheets to Databases Seamlessly

A PHP package that transforms spreadsheet data into database structures and content with minimal effort.

Overview

Derafu ETL provides a streamlined solution for converting data between spreadsheets and databases. With a clean, fluent API, it simplifies complex data integration tasks through a pipeline architecture.

```
$pipeline = new Pipeline();
$result = $pipeline
    ->extract('data.xlsx')      // Extract data from a spreadsheet.
    ->transform($rules)         // Apply transformations rules (optional).
    ->load('database.sqlite')   // Load into a database.
    ->execute()
;
```

Key Features

- **Extract** data from various sources (XLSX, ODS, CSV, databases).
- **Transform** data with customizable rules.
- **Load** data into different target systems.
- **Bidirectional** conversion between spreadsheets and databases.
- **Schema management** with automatic table creation and structure updates.
- **Data visualization** capabilities with schema export to Markdown, D2, and more.
- **Extensible** architecture for custom source and target systems.

Installation

Install via Composer:

```
composer require derafu/etl
```

Quick Start

Command Line

The quickest way to use Derafu ETL is through the command line:

```
php app/console.php derafu:etl data.xlsx database.sqlite
```

This extracts data from `data.xlsx` and loads it into a new SQLite database on `database.sqlite`.

Example

Run the example used in tests with:

```
php app/console.php derafu:etl tests/fixtures/spreadsheet-data.xlsx
```

This will create a `spreadsheet-data.sqlite` in the current directory.

PHP Code

```
use Derafu\ETL\Pipeline\Pipeline;

$pipeline = new Pipeline();
$result = $pipeline
    ->extract('data.xlsx')  // Load data from a XLSX.
    ->transform()           // This will use default transformations.
    ->load([                // You can specify the configuration for Doctrine.
        'doctrine' => [
            'driver' => 'pdo_sqlite',
            'path' => 'database.sqlite',
        ]
    ])
    ->execute();            // This is will run the process.

echo "Rows loaded: " . $result->rowsLoaded();
```

Understanding ETL Pipelines

An ETL pipeline consists of three main steps:

1. **Extract**: Read data from a source (e.g., spreadsheet).

2. **Transform**: Apply rules and transformations to the data.
3. **Load**: Write the transformed data to a target (e.g., database).

Derafu ETL provides a clean interface for each step while handling the complex details behind the scenes.

More than just move data to a target

Export Database Schema to Markdown

```
use Derafu\ETL\Database\DatabaseManager;
use Derafu\ETL\Schema\Target\MarkdownSchemaTarget;

$manager = new DatabaseManager();
$database = $manager->connect('database.sqlite');

$target = new MarkdownSchemaTarget();
$markdown = $target->applySchema($database->schema());

file_put_contents('schema.md', $markdown);
```

Generate Database Diagram

```
use Derafu\ETL\Database\DatabaseManager;
use Derafu\ETL\Schema\Target\D2SchemaTarget;

$manager = new DatabaseManager();
$database = $manager->connect('database.sqlite');

$target = new D2SchemaTarget();
$d2 = $target->applySchema($database->schema());

file_put_contents('schema.d2', $d2);
```

ETL Architecture

Architecture

Anonymous · 21/08/2026

Architecture

This document explains the architecture and design principles behind Derafu ETL.

ETL Pattern

The Extract-Transform-Load (ETL) pattern is a data integration process used to collect data from various sources, transform it to fit operational needs, and load it into a target database for analysis and storage.

The Three Phases

1. **Extract**: Gathering data from source systems.
2. **Transform**: Converting the extracted data to satisfy operational requirements.
3. **Load**: Writing the transformed data to the target system.

Derafu ETL Implementation

Derafu ETL implements this pattern with a clean, object-oriented approach centered around the Pipeline concept.

Core Components

Extract Phase

- `DataSource`: Encapsulates the data source (spreadsheet, database).
- `DataExtractor`: Handles extraction logic.
- `SchemaSource`: Extracts schema information from the source.

Transform Phase

- `DataRules`: Defines transformation rules.
- `DataTransformer`: Applies transformations to extracted data.

Load Phase

- `DataTarget`: Represents the destination system.
- `DataLoader`: Manages loading data into the target.
- `SchemaTarget`: Applies schema to the target system.

Pipeline Orchestration

The `Pipeline` class orchestrates the entire ETL process, providing a fluent interface:

```
$pipeline
    ->extract($source)    // Configure extraction.
    ->transform($rules)   // Configure transformation.
    ->load($target)       // Configure loading.
    ->execute()           // Run the pipeline.
;
```

When `execute()` is called, the pipeline:

1. Validates the configuration.
2. Extracts data from the source.
3. Transforms the data according to rules.
4. Synchronizes the target schema with the source.
5. Loads the transformed data into the target.
6. Returns a result object with statistics.

Key Abstractions

Database

The `Database` abstraction provides a unified interface for different database types:

- `SpreadsheetDatabase`: Treats spreadsheets as databases using Derafu Spreadsheet.
- `DoctrineDatabase`: Works with any database supported by Doctrine DBAL.

Schema

The `Schema` system represents database structure:

- Tables, columns, indexes, foreign keys.
- Import/export to various formats (Spreadsheet, Doctrine, Markdown, D2, etc.).

Extension Points

Derafu ETL is designed for extensibility:

1. **New Data Sources**: Implement `DataSourceInterface`.
2. **Custom Transformations**: Extend `DataRules`.
3. **New Data Targets**: Implement `DataTargetInterface`.
4. **Schema Visualization**: Implement `SchemaTargetInterface`.

Design Principles

1. **Separation of Concerns**: Each component has a clear responsibility.
2. **Fluent Interface**: Expressive, chainable API.
3. **Flexibility**: Support for various formats and systems.
4. **Extensibility**: Easy to extend with custom components.

Last updated on 21/08/2026

Schema Visualization

Schema Visualization

Anonymous · 21/08/2026

Schema Visualization

Derafu ETL includes powerful tools for visualizing database schemas through various output formats. These visualizations are useful for documentation, planning, and understanding the structure of your data.

Available Visualization Formats

Derafu ETL supports multiple visualization formats through schema targets:

- **Markdown**: Human-readable documentation.
- **D2**: Interactive entity-relationship diagrams.
- **Text**: Simple text representation.
- **SQL**: Database creation scripts.
- **Doctrine Schema**: Programmatic schema representation.

Using Schema Targets

Schema targets implement the `SchemaTargetInterface` and convert a schema to a specific format.

Basic Usage

All schema targets follow a similar pattern:

```
use Derafu\ETL\Database\DatabaseManager;
use Derafu\ETL\Schema\Target\CustomSchemaTarget; // Just an example.

// Connect to the database.
$manager = new DatabaseManager();
$database = $manager->connect('database.sqlite');

// Get the schema.
$schema = $database->schema();

// Create the target.
$target = new CustomSchemaTarget();
```

```
// Apply the schema to the target.
$output = $target->applySchema($schema);

// Use the output.
file_put_contents('output-file.custom', $output);
```

Markdown Schema

The `MarkdownSchemaTarget` generates comprehensive Markdown documentation for your database schema.

```
use Derafu\ETL\Schema\Target\MarkdownSchemaTarget;

$target = new MarkdownSchemaTarget();
$markdown = $target->applySchema($schema);

file_put_contents('schema.md', $markdown);
```

The generated Markdown includes:

- Table of contents.
- Detailed table definitions.
- Column information with types and constraints.
- Primary keys, indexes, and foreign key relationships.

Example output:

```
# Database Schema

## Table of Contents

- [Table: users](#table-users)
- [Table: posts](#table-posts)

## Table: users {#table-users}

### Columns

| Column     | Type        | Attributes              | Description |
|------------|-------------|-------------------------|-------------|
| id         | integer     | PRIMARY KEY / NOT NULL  |             |
| username   | string(100) | NOT NULL                |             |
| email      | string(255) | NOT NULL                |             |
| created_at | datetime    | NOT NULL                |             |

### Primary Key
```

```
- Columns: `id`

### Indexes

| Name         | Columns  | Type   | Flags |
|--------------|----------|--------|-------|
| idx_username | username | INDEX  |       |
| idx_email    | email    | UNIQUE |       |
```

D2 Diagrams

The `D2SchemaTarget` generates diagrams in D2 format, a modern diagram scripting language.

```
use Derafu\ETL\Schema\Target\D2SchemaTarget;

$target = new D2SchemaTarget(
    detailLevel: D2SchemaTarget::DETAIL_FULL,
    direction: D2SchemaTarget::DIRECTION_RIGHT,
    layout: D2SchemaTarget::LAYOUT_DEFAULT,
    includeIndexes: true
);
$d2 = $target->applySchema($schema);

file_put_contents('schema.d2', $d2);
```

Options include:

- **Detail Level**: `DETAIL_FULL`, `DETAIL_KEYS_ONLY`, `DETAIL_MINIMAL`.
- **Direction**: `DIRECTION_UP`, `DIRECTION_DOWN`, `DIRECTION_LEFT`, `DIRECTION_RIGHT`.
- **Layout**: `LAYOUT_DEFAULT`, `LAYOUT_CLUSTERED`, `LAYOUT_HIERARCHICAL`.
- **Include Indexes**: Whether to include indexes in the diagram.

Example D2 output:

```
# Database Schema

direction: right

# Tables
users: {
  shape: sql_table
  id: integer NOT NULL pk
  username: string(100) NOT NULL column
  email: string(255) NOT NULL column
  created_at: datetime NOT NULL column
}
```

```
posts: {
  shape: sql_table
  id: integer NOT NULL pk
  user_id: integer NOT NULL fk
  title: string(255) NOT NULL column
  content: text column
  created_at: datetime NOT NULL column
}

# Relationships
posts -> users
```

Practical Applications

Documentation

Generate comprehensive documentation for your database:

```
use Derafu\ETL\Database\DatabaseManager;
use Derafu\ETL\Schema\Target\MarkdownSchemaTarget;

$manager = new DatabaseManager();
$database = $manager->connect('production.sqlite');

$target = new MarkdownSchemaTarget();
$docs = $target->applySchema($database->schema());

file_put_contents('database-schema.md', $docs);
```

Visual Modeling

Create visual diagrams for presentations or analysis:

```
use Derafu\ETL\Schema\Target\D2SchemaTarget;

// Filter to only show specific tables.
$target = new D2SchemaTarget(
    tableFilter: ['users', 'posts', 'comments']
);

$diagram = $target->applySchema($schema);
file_put_contents('entity-relationship.d2', $diagram);
```

Schema Migration

Export a schema definition to create a new database:

```
use Derafu\ETL\Schema\Target\SqliteSchemaTarget;

$target = new SqliteSchemaTarget();
$sql = $target->applySchema($schema);

file_put_contents('schema.sql', $sql);
```

Integration with ETL Pipeline

Schema visualization can be integrated with the ETL pipeline for comprehensive documentation:

```
use Derafu\ETL\Pipeline\Pipeline;
use Derafu\ETL\Schema\Target\MarkdownSchemaTarget;

// Run ETL pipeline.
$pipeline = new Pipeline();
$result = $pipeline
    ->extract('data.xlsx')
    ->transform()
    ->load('database.sqlite')
    ->execute();

// Document the resulting database.
$database = $result->target()->database();
$target = new MarkdownSchemaTarget();
$docs = $target->applySchema($database->schema());

file_put_contents('database-schema.md', $docs);
```

Last updated on 21/08/2026