Introduction

Expressive Path-Based Query Builder for PHP

Anonymous · 09/09/2026

Expressive Path-Based Query Builder for PHP

`derafu/query` is a PHP library for building and filtering SQL queries using a compact, URL-safe string syntax. A single expression like `customers[alias:c]__invoices[on:id=customer_id,alias:i]__total?>1000` carries the full join path, join conditions, and filter in one string — no separate join calls required.

Why Derafu\Query?

Traditional query builders require explicit join definitions, verbose relationship navigation, and different filter syntaxes across engines. `derafu/query` replaces all of that with:

- A **single expression format** that encodes path, joins, and filter in one string.
- **Automatic join generation** from path segments — no manual `join()` calls when using path syntax.
- **50+ operators** covering comparisons, patterns, lists, ranges, dates, NULL, regex, bitwise, and subqueries.
- **Database-specific SQL** generated from the same expression — PostgreSQL, MySQL, and SQLite without code changes.
- **Framework bridges** for Doctrine DBAL, Doctrine ORM, Laravel's Illuminate query builder, and API Platform.

Feature	Derafu\Query	Traditional Query Builders
Path-Based Relationships	✅	❌
Automatic Join Resolution	✅	❌
Configurable Operators	✅	❌
Framework Agnostic Core	✅	⚠
Unified Filter Syntax	✅	⚠
Multi-DB SQL Generation	✅	⚠

Installation

```
composer require derafu/query
```

Quick Start

With SqlQueryBuilder (standalone)

```
use Derafu\Query\Builder\SqlQueryBuilder;
use Derafu\Query\Engine\PdoEngine;
use Derafu\Query\Filter\ExpressionParser;
use Derafu\Query\Filter\PathParser;
use Derafu\Query\Filter\FilterParser;
use Derafu\Query\Filter\CompositeExpressionParser;
use Derafu\Query\Operator\OperatorLoader;
use Derafu\Query\Operator\OperatorManager;

$pdo    = new PDO('sqlite:/path/to/db.sqlite');
$engine = new PdoEngine($pdo);

$loader  = new OperatorLoader();
$manager = new
OperatorManager($loader->loadFromFile('vendor/derafu/query/resources/operators.yaml'))
;
$parser  = new CompositeExpressionParser(
    new ExpressionParser(new PathParser(), new FilterParser($manager))
);

$qb = new SqlQueryBuilder($engine, $parser);

// Simple filter.
$rows = $qb->table('products')->where('price?>1000')->execute();

// Multi-table path (auto-generates the JOIN).
$rows = $qb
    ->select('c.name AS customer_name, i.number, i.total')
    ->where('customers[alias:c]__invoices[on:id=customer_id,alias:i]__total?>1000')
    ->execute();
```

With a Framework Bridge

```
// Doctrine DBAL.
```

```
use Derafu\Query\Bridge\DoctrineDBALQueryBuilderConditionApplier;
use Derafu\Query\Filter\CompositeExpressionParser;

$applier = new DoctrineDBALQueryBuilderConditionApplier();
$condition = $parser->parse('status?=active&&total?>1000');
$applier->apply($dbalQueryBuilder, $condition);
```

The Expression Format

Every filter is a string of the form:

```
path?filter
```

The `?` separates the **path** (which column or relationship to target) from the **filter** (which operator and value to apply).

```
price?>1000                          ← column "price", operator ">", value "1000"
status?in:paid,issued                ← column "status", operator "in:", value
"paid,issued"
created_at?date:20240301             ← column "created_at", operator "date:", value
"20240301"
deleted_at?is:null                   ← column "deleted_at", operator "is:null", no
value
```

Multi-segment paths navigate relationships and generate joins automatically:

```
customers[alias:c]__invoices[on:id=customer_id,alias:i]__total?>1000
```

Subquery paths starting with `___` generate correlated EXISTS / aggregate subqueries:

```
___payments[on:id=invoice_id]?is:empty          ← NOT EXISTS (payments)
___payments[on:id=invoice_id]__status?=pending  ← EXISTS with column filter
___payments[on:id=invoice_id]__SUM(amount)?>=500← aggregate scalar subquery
```

Multiple conditions can be combined with `&&` (AND), `||` (OR), and `()` (grouping):

```
status?=active&&total?>1000
category?=electronics||(category?=software&&price?<500)
```

Operator Overview

`derafu/query` ships with over 50 operators across 10 types:

Type	Examples	SQL produced
Standard	`=`, `!=`, `>`, `<`, `>=`, `<=`	`col = :p`, `col > :p`
AutoLike	`^`, `$`, `~~`, `~~*`, `!~~`	`col LIKE :p` with auto `%`
Like	`like:`, `ilike:`, `notlike:`	`col LIKE :p`, `col ILIKE :p`
List	`in:`, `notin:`	`col IN (:p1, :p2, …)`
Range	`between:`, `notbetween:`	`col BETWEEN :p1 AND :p2`
Date	`date:`, `month:`, `year:`, `period:`	`DATE(col) = :p`, etc.
NULL	`is:null`, `isnot:null`, `<=>`	`col IS NULL`
Subquery	`is:empty`, `isnot:empty`	`NOT EXISTS (…)`, `EXISTS (…)`
RegExp	`~`, `~*`, `!~`, `similarto:`	`col ~ :p`, `col SIMILAR TO :p`
Binary	`b&`, `b|`, `b^`, `b<<`, `b>>`	`col & :p`, etc.

What This Documentation Covers

Page	Content
Architecture	Layer structure, class responsibilities, data flow
Expression Syntax	`path?filter` format, composite `&&`/`||`/`()`
Path Syntax	Segments, options, join paths, subquery paths
Operators Reference	All operators with SQL templates, validation, casting
Query Builder	Fluent API and declarative `QueryConfig`
Framework Bridges	Doctrine DBAL/ORM, Illuminate, API Platform
Security Guide	SQL sanitization and safe usage patterns

Last updated on 09/09/2026