django_prepared_query Documentation

Release 0.1

Dima Kudosh

Jul 16, 2018

Contents

SQL prepared statement support for Django ORM.

CHAPTER 1

Installation

django_prepared_query works only with Python 3 and Django 1.11+. Currently it supports only PostgreSQL and MySQL. It can be installed with **pip**:

```
$ pip install django_prepared_query
```

CHAPTER 2

Usage

For using prepared statements you must replace model standard manager with *PreparedManager*.

```
from django_prepared_query import PreparedManager

class Book(Model):
    objects = PreparedManger()
    ...
```

PreparedManager has 2 additional methods: prepare and execute that are equivalent to SQL prepare and execute commands.

```
qs = Book.objects.prepare()
result = qs.execute()
```

Note: After prepare you can call only execute method, other methods raises *OperationOnPreparedStatement* exception. Calling execute before prepare raises *QueryNotPrepared* exception.

It's possible to add dynamic parameters to query using *BindParam* expression. It takes parameter name and not required field type for situations when it can't automatically detect type. Parameter name will be used in execute method. Passing incorrect parameter name raises *IncorrectBindParameter* exception.

```
from django_prepared_query import BindParam

qs = Book.objects.filter(name=BindParam('book_name')).prepare()
result = qs.execute(book_name='Harry Potter')
```

Also you can use different built-in lookups except *isnull* that raises *NotSupportedLookup* exception.

```
from django_prepared_query import BindParam

qs = Book.objects.filter(name__startswith=BindParam('book_name')).prepare()
```

(continues on next page)

(continued from previous page)

```
result = qs.execute(book_name='Harry Potter')
result = qs.execute_iterator(book_name='Harry Potter')  # Returns iterator
```

Before running execute query django_prepared_query validates input parameter types, *ValidationError* will be raised in cases when parameter type isn't matched.

BindParam can be used in queryset slicing as well.

```
qs = Book.objects.all()[BindParam('start'):BindParam('end')].prepare()
result = qs.execute(start=0, end=20)
```

Also you can create prepared quires with in lookup. For this you should use *BindArray* expression instead of *BindParam*. *BindArray* accepts additional parameter size, so you can use only arrays with specified size or smaller. In cases when you pass smaller array to *BindArray* expression, it will fill remaining positions with *NULL* that will pe optimized by database.

```
from django_prepared_query import BindArray

qs = Book.objects.filter(id__in=BindArray('ids', 10)).prepare()
result = qs.execute(ids=list(range(10)))
```

CHAPTER 3

Contributing

To contribute to django_prepared_query create a fork on GitHub. Clone your fork, make some changes, and submit a pull request. Issues

CHAPTER 4

Issues

Use the GitHub issue tracker to submit bugs, issues, and feature requests.