



DaneSoul 8 мая 2020 в 04:45

## Python: Работа с базой данных, часть 2/2: Используем ORM

Python\*, Программирование\*, SQL\*, SQLite\*

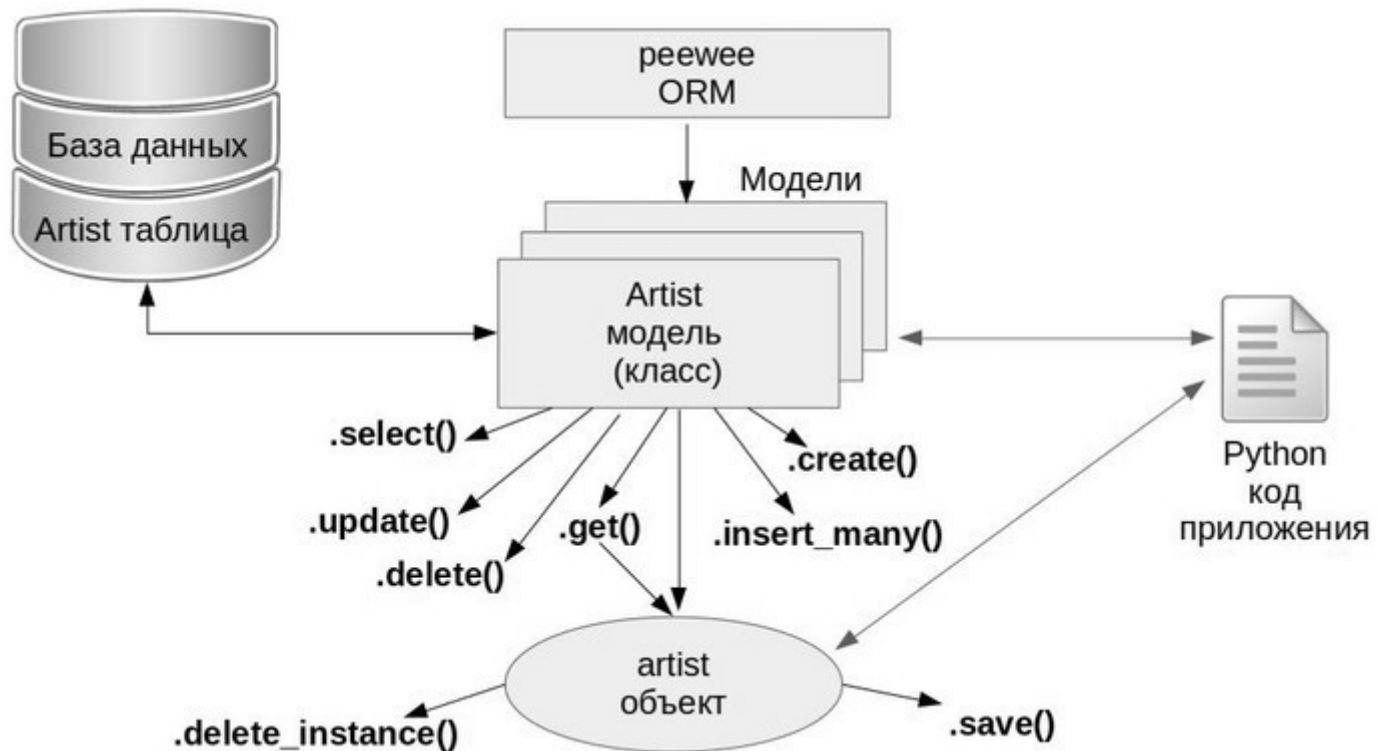
Tutorial

часть 1/2: Используем DB-API

часть 2/2: Используем ORM

Это вторая часть моей статьи по работе с базой данных в Python. В первой части мы рассмотрели основные принципы коммуникации с SQL базой данных, а в этой познакомимся с инструментарием, позволяющим облегчить нам это взаимодействие и сократить количество нашего кода в типовых задачах.

Статья ориентирована в первую очередь на начинающих, она не претендует на исчерпывающе глубокое изложение, а скорее дает краткую вводную в тему, объясняет самые востребованные подходы для старта и иллюстрирует это простыми примерами базовых операций.



Требуемый уровень подготовки: базовое понимание SQL и Python (код статьи проверялся под Python 3.6). Желательно ознакомиться с первой частью, так как к ней будут неоднократные отсылки и сравнения. В конце статьи есть весь код примеров под спойлером в едином файле и список ссылок для более углубленного изучения материала.

### 1. Общие понятия ORM

В нашем коде мы работаем с объектами разной природы, а при работе с SQL базой данных мы вынуждены постоянно генерировать текстовые запросы к базе, а получив ответ от базы обратно его преобразовывать в формат данных нашего приложения.

Хорошо было бы иметь некий механизм автоматического генерирования этих запросов исходя из заранее определенной структуры наших данных и приведения ответа к этой же структуре. Именно таким механизмом является добавление дополнительной ORM-прослойки между кодом нашего приложения и SQL базой.

В случае высоко нагруженных проектов использование такой прослойки может вызывать дополнительный расход ресурсов и требовать тонкой настройки, но это выходит за рамки нашей статьи.

Существуют два основных подхода к реализации ORM:

**Active Record** – более простой для понимания и реализации подход, суть которого сводится к отображению объекта данных на строку базы данных.

**Data Mapper** – в отличии от предыдущего подхода полностью разделяет представление данных в программе от его представления в базе данных.

У обоих подходов есть свои особенности, преимущества и недостатки, в зависимости от того, какого типа приложение Вы разрабатываете. В конце статьи есть несколько ссылок на статьи в которых подробно сравниваются эти два подхода с примерами.

В данном руководстве будет проиллюстрирован более простой и понятный для старта подход Active Record. Мы будем рассматривать основы работы с рееее – лёгкой, быстрой, гибкой ORM на Python, которая поддерживает SQLite, MySQL и PostgreSQL.

## Безопасность и SQL-инъекции

По умолчанию рееее будет параметризовать запросы, поэтому любые параметры, передаваемые пользователем, будут экранированы и безопасны.

Единственное исключение этому правилу это передаваемые прямые SQL запросы, передаваемые в SQL объект, которые могут содержать небезопасные данные. Для защиты от такой уязвимости, передавайте данные как параметры запроса, а не как часть SQL запроса. Тема экранирования данных обсуждалась в первой части статьи.

## 2. Установка ORM, соединение с базой, получение курсора

В качестве тестовой базы данных будем использовать ту же самую тестовую Chinook SQLite базу, что и в первой части статьи, там также в первой части есть примеры средств для наглядного просмотра содержимого этой базы.

В отличии от модуля sqlite из стандартной библиотеки, рееее прежде чем импортировать надо установить:

```
pip install peewee
```

Для начала рассмотрим самый базовый шаблон DB-API, который будем использовать во всех дальнейших примерах:

```
# Импортируем библиотеку, соответствующую типу нашей базы данных
# В данном случае импортируем все ее содержимое, чтобы при обращении не писать кажды
from peewee import *

# Создаем соединение с нашей базой данных
# В нашем примере у нас это просто файл базы
conn = SqliteDatabase('Chinook_Sqlite.sqlite')

# ТУТ БУДЕТ КОД НАШИХ МОДЕЛЕЙ

# Создаем курсор - специальный объект для запросов и получения данных с базы
cursor = conn.cursor()

# ТУТ БУДЕТ НАШ КОД РАБОТЫ С БАЗОЙ ДАННЫХ

# Не забываем закрыть соединение с базой данных
conn.close()
```

Собственно говоря, этот шаблон крайне похож на тот, который мы использовали в первой статье, отличие в методе соединения с базой данных. Теперь мы подключаемся через метод библиотеки peewee:

```
conn = SqliteDatabase('Chinook_Sqlite.sqlite')
```

В зависимости от типа нашей базы методы подключения отличаются: `SqliteDatabase()`, `MySQLDatabase()`, `PostgresqlDatabase()` — какой для какой базы очевидно из имени, в скобках передаются параметры подключения, в нашем примере это просто имя файла базы.

Обратите внимание, подключение ORM в данном случае не отбирает возможность использовать курсор для обычных запросов к базе, как мы делали в первой статье. При этом ORM выполняет функцию драйвера базы и нет необходимости дополнительно импортировать отдельно модуль `sqlite`.

То есть, мы можем взять наш новый шаблон, вставить в него код из первой статьи и получить ровно тот же результат:

```
# Делаем SELECT запрос к базе данных, используя обычный SQL-синтаксис
cursor.execute("SELECT Name FROM Artist ORDER BY Name LIMIT 3")
```

```
# Получаем результат сделанного запроса
results = cursor.fetchall()
print(results)  # [('A Cor Do Som',), ('AC/DC',), ('Aaron Copland & London Symphony
```

Это может быть очень удобно при постепенном переходе на ORM, так как мы можем сменить способ соединения с базой на работу через реееее и потом постепенно менять запросы к базе на новые, не нарушая работу старого кода!

### 3. Описание моделей и их связь с базой данных

Для работы с нашими данными через ORM мы для начала должны описать модели наших данных, чтобы построить связь между базой и объектами данных в нашем приложении.

Классы моделей, поля экземпляров и экземпляры моделей реееее соответствуют следующим концепциям базы данных:

ORM концепция	Концепция базы данных
Класс модели	Таблица базы данных
Поле экземпляра (атрибут объекта)	Колонка в таблице базы данных
Экземпляр модели (объект)	Строка в таблице базы данных

Для реальных проектов, имеет смысл вынести модели в отдельный файл или файлы, в нашем упрощенном учебном примере мы просто вставляем код определения модели сразу после строки соединения с базой данных, вместо строки **# ТУТ БУДЕТ КОД НАШИХ МОДЕЛЕЙ**

```
# Определяем базовую модель о которой будут наследоваться остальные
class BaseModel(Model):
    class Meta:
        database = conn # соединение с базой, из шаблона выше

# Определяем модель исполнителя
class Artist(BaseModel):
    artist_id = AutoField(column_name='ArtistId')
    name = TextField(column_name='Name', null=True)

    class Meta:
        table_name = 'Artist'
```

В данной статье не будем заострять внимание на различные типы полей и их связь с типами данных в различных базах данных. В документации к реееее есть детальная таблица связи между типом поля в нашей модели и в базе данных.

Обратите внимание, что требования сразу задать модели для всех таблиц нет. То есть в нашей тестовой базе есть несколько таблиц, но для наших примеров мы сейчас опишем только одну и будем дальше с ней работать, не трогая остальные. Таким образом, можно постепенно переводить код на ORM, не нарушая работу старого кода.

Замечание: Есть возможность автоматической генерации моделей из существующей базы данных, а также возможность генерации таблиц базы данных из заранее определенных моделей. Для подобных задач в реестре есть набор инструментария, так называемый Playhouse.

## 4. CRUD операции и общие подходы

Ниже мы рассмотрим так называемые CRUD операции с базой – создание (Create), чтение (Read), обновление (Update) и удаления (Delete) объектов/записей в базе. Мы не будем пытаться охватить все многообразие возможностей которые предоставляет нам реестр, рассмотрим только самые базовые вещи, которые позволят нам начать решать реальные задачи разработки. Более детальные описания можно найти в официальной документации.

Есть два основных подхода при работе с ORM реестра, в зависимости от того, какую задачу мы решаем и как нам удобней это делать:

1) Мы можем вызывать общие методы у класса модели, такие как `.select()`, `.update()`, `.delete()`, `.create()` и т.д., передавать дополнительные параметры и делать массовые операции. В данном случае, логика нашей работы похожа на логику работы с SQL запросами, которую мы рассматривали в первой статье. Основное отличие в том, что работая через модели у нас уже есть привязки к таблицам и известны имеющиеся поля, поэтому нам не надо это все явно прописывать в запросе.

2) Второй подход, состоит в том, что мы получаем объект класса модели, который соответствует одной строке таблицы базы данных, работаем с этим объектом, в том числе меняя значения его атрибутов, а по завершению работы сохраняем / обновляем — `.save()` или удаляем строку его представления в таблице базы данных — `.delete_instance()`.

Как это работает будем понятней из примеров CRUD операций ниже.

## 5. Чтение записей

### 5.1) Получение одиночной записи с методом модели `Model.get()`

```
artist = Artist.get(Artist.artist_id == 1)
```

Теперь у нас есть объект `artist`, с полями соответствующим данным исполнителя в конкретной строке, а также доступными методами модели исполнителя.

Этот объект можно использовать не только для чтения данных, но и для их обновления и удаления данной записи, в чем убедимся позже.

```
print('artist: ', artist.artist_id, artist.name) # artist: 1 AC/DC
```

## 5.2) Получение набора записей через нашу модель `Model.select()`

Это похоже на стандартный select запрос к базе, но осуществляемый через нашу модель.

Обратите внимание, что к какой таблице обращаться и какие поля у нее есть

уже определено в нашей модели и нам не надо это указывать в нашем запросе.

Формируем запрос к базе с помощью нашей ORM прослойки и смотрим как этот запрос будет выглядеть:

```
query = Artist.select()
print(query)
# SELECT "t1"."ArtistId", "t1"."Name" FROM "Artist" AS "t1"
```

Полезно добавить дополнительные параметры, уточняющие запрос, они очень похожи на SQL инструкции:

```
query = Artist.select().where(Artist.artist_id < 10).limit(5).order_by(Artist.artist_id)
print(query)
# SELECT "t1"."ArtistId", "t1"."Name" FROM "Artist" AS "t1" WHERE ("t1"."ArtistId" < 10) ORDER BY "t1"."ArtistId" ASC
```

Теперь, определившись с запросом к базе, мы можем получить от нее ответ, для удобства делаем это сразу в виде словаря

```
artists_selected = query.dicts().execute()
print(artists_selected) # <peewee.ModelDictCursorWrapper object at 0x7f6fdd9bdda0>
```

Мы получили итератор по полученным из базы записям, который можно обходить в цикле `for artist in artists_selected` и получать сразу словари, соответствующие структуре нашего исполнителя, каждая итерация соответствует одной строке таблицы и соответственно одному исполнителю:

```
for artist in artists_selected:
    print('artist: ', artist) # artist: {'artist_id': 9, 'name': 'BackBeat'}
```

Для упрощения дальнейшей визуализации изменений в базе при дальнейших наших операциях добавим в наш шаблон под определением моделей код следующей функции:

```
def print_last_five_artists():
    """ Печатаем последние 5 записей в таблице исполнителей """
    print('#####')
    cur_query = Artist.select().limit(5).order_by(Artist.artist_id.desc())
    for item in cur_query.dicts().execute():
        print('artist: ', item)
```

Из кода достаточно очевидно, что функция просто выводит на печать 5 последних записей исполнителей с базы, что позволит нам видеть какие данные добавились, обновились или удалились в примерах ниже.

Обращаем внимание, что вывод будет совпадать с примерами в статье, только если их выполнять последовательно, начиная с неизменной Chinook базы, так как как примеры модифицируют базу!

## 6. Создание записи

**6.1) Первый способ:** `Model.create()` — передаем все требуемые параметры сразу

```
Artist.create(name='1-Qwerty')
```

**6.2) Второй способ:** Мы создаем объект класса нашей модели, работаем в коде в содержимом его полей, а в конце вызываем его метод `.save()`

```
artist = Artist(name='2-asdfg')
artist.save()
```

Обратите внимание, что здесь метод вызываем у объекта класса модели, а не у самой модели, как в первом способе.

**6.3) Третий способ** — массовое добавление из коллекции методом модели `Model.insert_many()`

Обратите внимание, что первые два метода не требуют добавления `.execute()`, а этот требует!

```
artists_data = [{'name': '3-qaswed'}, {'name': '4-yhnbgt'}]
Artist.insert_many(artists_data).execute()
```

Визуализируем последние 5 записей в таблице исполнителей, чтобы убедиться, что три примера выше доавили нам 4 новые записи:

► [`print\_last\_five\_artists\(\)`](#)

## 7. Обновление записей

**7.1) Первый способ обновления записей.**

Выше, способом 6.2 мы создавали новую запись, но так можно не только создавать новую запись, но и обновлять существующую. Для этого нам надо для нашего объекта указать уже существующий в таблице первичный ключ.

```
artist = Artist(name='2-asdfg+++++')
artist.artist_id = 277 # Тот самый первичный ключ
# который связывает наш объект с конкретной строке таблицы базы данных
artist.save()
```

## ► [print\\_last\\_five\\_artists\(\)](#)

**7.2) Для обновления многих записей сразу**, можно использовать метод модели `Model.update()`, в котором указываем что именно у нас меняется, а метод `.where()` определяет по каким критериям отбираются записи для изменения

```
query = Artist.update(name=Artist.name + '!!!!').where(Artist.artist_id > 275)
query.execute()
```

## ► [print\\_last\\_five\\_artists\(\)](#)

## 8. Удаление записей

**8.1) Первый способ удаления записи** — это получение объекта записи методом `Model.get()` как в 5.1 выше и вызова метода удаления этой записи `.delete_instance()`:

```
artist = Artist.get(Artist.artist_id == 279)
artist.delete_instance()
```

## ► [print\\_last\\_five\\_artists\(\)](#)

**8.2) Для удаления набора строк** можно использовать `Model.delete()` метод

```
query = Artist.delete().where(Artist.artist_id > 275)
query.execute()
```

## ► [print\\_last\\_five\\_artists\(\)](#)

## 9. Полный код всех примеров в шаблоне с комментариями

## ► [Показать полный код всех примеров и шаблона](#)

## 10. Список литературы

- The Active Record and Data Mappers of ORM Pattern (на английском)
- What's the difference between Active Record and Data Mapper? (на английском)
- Active Record против Data Mapper-а для сохранения данных
- Chinook — Sample database for SQL Server, Oracle, MySQL, PostgreSQL, SQLite, DB2
- реееее официальные страницы репозитория и документации (на английском)
- Peewee tutorial (на английском)
- Peewee – лёгкая, гибкая и очень быстрая ORM на Python



## Приглашаю к обсуждению:

- Если я где-то допустил неточность или не учёл что-то важное — пишите в комментариях, важные комментарии будут позже добавлены в статью с указанием вашего авторства.
- Если какие-то моменты не понятны и требуется уточнение — пишите ваши вопросы в комментариях — или я или другие читатели дадут ответ, а дельные вопросы с ответами будут позже добавлены в статью.

**Теги:** Python, SQL, ORM, peewee, SQLite, база данных, db

**Хабы:** Python, Программирование, SQL, SQLite

## Редакторский дайджест

Присылаем лучшие статьи раз в месяц

Электронпочта



64

2

Карма Рейтинг

**Александр @DaneSoul**

Веб-программирование, Python

Задонатить

## Комментарии 15



**stepalxser**

08.05.2020 в 19:55

~~Природа настолько отчистилась за время карантина, что даже автор продолжил топик через 3.5 года.~~ А так спасибо за все материалы, будут ли дальше какие-либо статьи по orm в питоне? Может сравнение актуальных?



0

Ответить



**DaneSoul**

08.05.2020 в 19:57

Как говорится, лучше поздно, чем никогда.

По ORM и базам данных больше статей пока не планирую, а там время покажет.



0


Ответить



 **InBioReactor**  
22.03.2021 в 13:59

Я веду свои биологические записи в excel и все это надо будет потом подготовить и перевести в базу данных, проблема в том, что в настоящих базах данных я не работал. Хотелось бы узнать о таблицах больше

 0 Ответить

 **DaneSoul**  
22.03.2021 в 15:10

Из Excel данные можно сохранить в виде CSV файла (текстовый формат с разделителями), а потом с помощью Python считать этот CSV (есть функции в стандартной библиотеке) и сохранить в базу. Это достаточно стандартные вещи, которым посвящено масса материалов в сети.

 0 Ответить

 **sled**  
12.08.2021 в 21:55

Можете продолжить работу с данными в Excel, а обработку, структурирование и анализ делать программно тем же Python-ом. Для обращения к данным в файлах Excel имеются питоновские библиотеки, задав в поисковике - "python excel" Вы о них узнаете. Например: xlrd, xlwt, xlutils, openpyxl, pandas, pyexcel.

Можно подключаться непосредственно к приложению Excel, как к COM-объекту - <https://habr.com/ru/post/232291/#com>

 0 Ответить

 **macik**  
12.01.2021 в 00:03

Все вроде отлично, но с Mysql завести не могу вообще.

PS

Я новичок в Python

 0 Ответить

 **DaneSoul**  
12.01.2021 в 00:10

Все вроде отлично, но с Mysql завести не могу вообще.

Прежде всего убедитесь, что у Вас правильно настроено соединение с базой.

```
conn = SQLiteDatabase('Chinook_Sqlite.sqlite')
```

Вместо этой строки из примера должно быть соединение с Вашей базой.

Вот тут например посмотрите как оно делается.

 0 Ответить

 **macik**  
12.01.2021 в 03:06

Я попробовал два варианта

```
db_204 = mysql.connector.connect(  
host = '192.168.22.204',  
user = '1',  
password = '1',  
database = 'user'  
)  
print(db_204)
```

Ответ <mysql.connector.connection\_cext.CMySQLConnection object at 0x7f95a0063580>

```
db_204 = MySQLDatabase(  
host = '192.168.22.204',  
user = '1',  
password = '1',  
database = 'user'  
)
```

Ответ <peewee.MySQLDatabase object at 0x7fb4b0063820>

Потом попробовал...

```
class BaseModel(Model):  
class Meta:  
database = db_204 # соединение с базой, из шаблона выше
```

```
class Firms(BaseModel):  
firm_id = AutoField(column_name='id')  
name = TextField(column_name='name', null=True)
```

```
class Meta:  
table_name = 'firms'
```

```
query = Firms.select().where(Firms.firm_id <  
10).limit(5).order_by(Firms.firm_id.desc())  
firms_selected = query.dicts().execute()
```

Сам запрос получается корректным, я проверял на phpmyadmin, он выдает то что надо. Но вот в Python выдает ошибку

```
Traceback (most recent call last):  
File "peewee_my.py", line 33, in firms_selected = query.dicts().execute()  
File "/Users/maciborka/Library/Python/3.8/lib/python/site-packages/peewee.py",  
line 1898, in inner  
return method(self, database, *args, **kwargs)  
File "/Users/maciborka/Library/Python/3.8/lib/python/site-packages/peewee.py",  
line 1969, in execute  
return self._execute(database)  
File "/Users/maciborka/Library/Python/3.8/lib/python/site-packages/peewee.py",  
line 2141, in _execute
```

```
cursor = database.execute(self)
File "/Users/maciborka/Library/Python/3.8/lib/python/site-packages/peewee.py",
line 3142, in execute
return self.execute_sql(sql, params, commit=commit)
File "/Users/maciborka/Library/Python/3.8/lib/python/site-packages/peewee.py",
line 3127, in execute_sql
cursor = self.cursor(commit)
File "/Users/maciborka/Library/Python/3.8/lib/python/site-packages/peewee.py",
line 3111, in cursor
self.connect()
File "/Users/maciborka/Library/Python/3.8/lib/python/site-packages/peewee.py",
line 3065, in connect
self._state.set_connection(self._connect())
File "/Users/maciborka/Library/Python/3.8/lib/python/site-packages/peewee.py",
line 3964, in _connect
raise ImproperlyConfigured('MySQL driver not installed!')
peewee.ImproperlyConfigured: MySQL driver not installed!
```

0 Ответить



macik

12.01.2021 в 03:17

Нашел причину. Надо было поставить `pip install mysqlclient`

Сложно с этими библиотеками, только с опытом будет понятно что и как.

Можете подсказать, как все библиотеки четко в конфиге держать, что бы потом на любом сервере быстро развернуть?

Спасибо.

0 Ответить



DaneSoul

12.01.2021 в 10:05

Можете подсказать, как все библиотеки четко в конфиге держать, что бы потом на любом сервере быстро развернуть?

Для этого используется виртуальное окружение `venv`.

0 Ответить



macik

16.01.2021 в 07:05

Спасибо, за ответ.

Я не про это...

Я нашел что я хотел. Это файл с описанием всех пакетов которые вносятся в проект. Потом на любом сервере одной командой все зависимости ставятся.

```
pip install -r requirements.txt
```

0 Ответить



**DaneSoul**  
16.01.2021 в 14:28

Это именно про это.

Потому как если у Вас в одном окружении все проекты, то данная команда будет тащить ВСЕ установленные модули. А если Вы для каждого проекта создадите свое виртуальное окружение, то модули будут изолированы и можно будет сохранять/переносить только актуальные для данного проекта модули.

0 Ответить



**DaneSoul**  
16.01.2021 в 18:26

Точнее не эта команда будет тащить все модули, а команда записи зависимостей в файл, с которого потом их будете читать.

0 Ответить



**macik**  
16.01.2021 в 22:57

Все что будет в файле обычно это то что надо для проекта. У каждого проекта свой файл.

0 Ответить



**DollaR84**  
12.01.2021 в 08:00

Если использовать стандартный класс:

```
from peewee import MySQLDatabase
```

то у меня возникала точно такая же ошибка. Но можно не устанавливать никаких дополнительных библиотек, а использовать расширенный класс из стандартного пакета реееее:

```
from playhouse.mysql_ext import MySQLConnectorDatabase
```

Данный класс нормально подключается и работает без установки чего-либо дополнительного.

0 Ответить

Только полноправные пользователи могут оставлять комментарии. [Войдите](#), пожалуйста.

## Публикации

ЛУЧШИЕ ЗА СУТКИ

ПОХОЖИЕ



alexopryshko сегодня в 17:58

## Школу закончил в 14, Бауманку в 18: почему, зачем и какие последствия

◆ +41

👁 8.1K

📖 20

💬 57 +57



MaFrance351 сегодня в 15:01

## Считываем и эмулируем карты с магнитной полосой

◆ +40

👁 3.3K

📖 28

💬 19 +19



elena\_pastukhova сегодня в 19:01

## Как мы съедаем 15 тонн воды в день

◆ +29

👁 3.6K

📖 7

💬 12 +12



Sagidullin сегодня в 03:53

## Big bada boom отменяется? Подводные интернет-магистралы выдержат наступление «События Кэррингтона»

◆ +29

👁 2.5K

📖 5

💬 12 +12



sergey\_\_pushkin сегодня в 14:01

## Язык диаграмм

◆ +27

👁 2.3K

📖 26

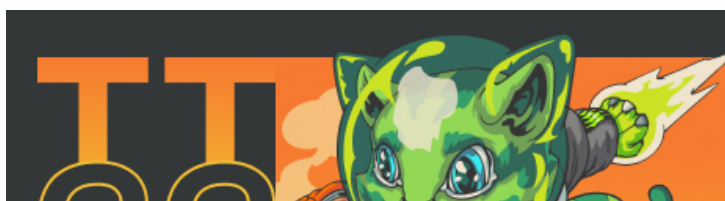
💬 7 +7

## Небольшой проект на Go, до 1000 CPU: в четыре шага определяем, какие мощности нужны сервису

Турбо

МИНУТОЧКУ ВНИМАНИЯ

Разместить





Нужен ваш лучший пост за год: с экспертизой и котами



Быстрый и производительный: iOS против Android

## КУРСЫ



Аналитик данных

5 декабря 2022 · 100 375 ₽ · Нетология



Python-разработчик с нуля

8 декабря 2022 · 90 750 ₽ · Нетология



Углубленный курс по Python

10 декабря 2022 · 45 000 ₽ · GB



SQL и получение данных

19 декабря 2022 · 24 850 ₽ · Нетология



Основы программирования на Python. Уровень 2

22 декабря 2022 · 19 500 ₽ · Level UP

Больше курсов на Хабр Карьере

## ЧИТАЮТ СЕЙЧАС

Как школьники МЭШ взломали



34K



69 +69

Школу закончил в 14, Бауманку в 18: почему, зачем и какие последствия



8K



57 +57

Ловушка алгоритмизации, или как 44-ФЗ породил коррупцию



2.3K



36 +36

Lumia 640 — всё ещё достоин?



1.9K



7 +7

Как мы съедаем 15 тонн воды в день



2.6K




12 +12

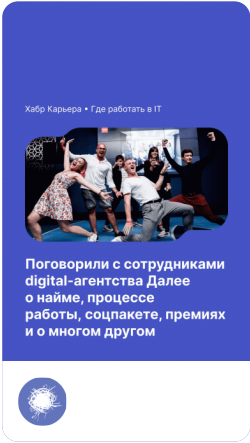
Итоги сезона Data Mining: тем много, но в топе NLP и гайды

Мегапост


ИСТОРИИ



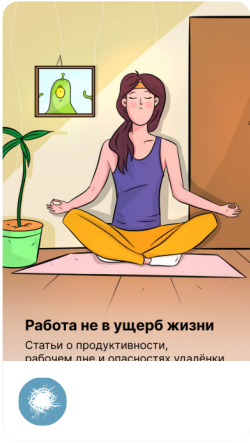
Причины возгорания аккумулятора



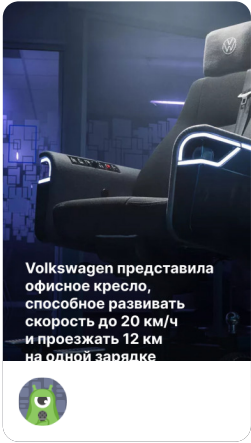
Где работать в IT: Далее



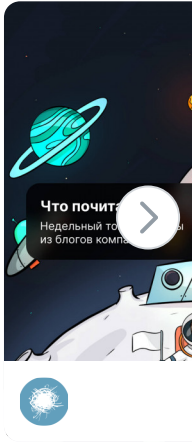
Итоги сезона Data Mining



Не работай через силу



Офисное кресло Volkswagen



Недельный топ годноты от компаний

РАБОТА

- Python разработчик  
164 вакансии
- Data Scientist  
122 вакансии
- Django разработчик  
49 вакансий

Все вакансии

Ваш аккаунт

- Войти
- Регистрация

Разделы

- Публикации
- Новости
- Хабы

Информация

- Устройство сайта
- Для авторов
- Для компаний

Услуги

- Корпоративный блог
- Медийная реклама
- Нативные проекты



[Компании](#)

[Документы](#)

[Авторы](#)

[Соглашение](#)

[Образовательные](#)

[Песочница](#)

[Конфиденциальность](#)

[программы](#)

[Стартапам](#)

[Мегaproекты](#)



[Настройка языка](#)

[Техническая поддержка](#)

[Вернуться на старую версию](#)

© 2006–2022, Habr