Peewee ORM манипуляция базами данных

Ни для кого не новость, что большинство сегодняшних приложений взаимодействуют с базами данных. Особенно с движками на основе **RDBMS** (движки DB с поддержкой SQL). Как и любой другой язык программирования, Pyhton также предоставляет как собственные библиотеки для взаимодействия с базами данных, так и от третьих лиц. Как правило, вам нужно прописать запросы SQL для CRUD операций. Это нормально, однако иногда получается мешанина:

> Шеф решил перейти с MySQL в… MSSQL и у вас нет выбора, кроме как кивнуть и внести правки в свои запросы в соответствии с другим движком баз данных;
>
> Вам нужно сделать несколько запросов, чтобы получить одну часть данных из другой таблицы;
>
> Список можно продолжать долго, не так ли?

Чтобы разобраться с этим, в игру вступает **ORM**.

Введение в ORM

ORM – это акроним от **Object Relational Mapping** (Объектно-реляционное отображение). Но что именно оно делает?

Из **википедии**:

> **Объектно-реляционное** отображение — это технология программирования, которая связывает базы данных с концепциями **объектно-ориентированных**

языков программирования, создавая «виртуальную объектную базу данных». Существуют как проприетарные, так и свободные реализации этой технологии. Звучит круто, да?

Что такое Peewee?

Peewee (http://docs.peewee-orm.com/en/latest/) – это небольшое ORM, которое в данный момент поддерживает **postgresql**, **mysql** и sqlite. Разумеется, это не единственное ORM для разработчиков Python. К примеру, Django предоставляет собственную ORM библиотеку, кроме этого, всегда есть **SqlAlchemy**. Хорошая сторона **Peewee** – это то, что он занимает мало места, его легко освоить, и вы можете приступить к работе с приложениями за несколько минут.

Достаточно слов, перейдем к кодам!

Установка Peewee

Как и многие другие библиотеки Python, вы можете установить Peewee при помощи pip:

```Shell
pip install peewee
```

Настройка базы данных

Как я говорил, **Peewee** поддерживает ряд движков для работы с базами данных (полный список тут: http://docs.peewee-orm.com/en/latest/peewee/database.html), в данной статье я использую **MySQL**.

```Python
from peewee import *

user = 'root'
password = 'root'
db_name = 'peewee_demo'

dbhandle = MySQLDatabase(
    db_name, user=user,
    password=password,
    host='localhost'
)
```

Объект **MySQLDatabase** создан.

Создание моделей в Peewee

Сейчас я перейду к **созданию моделей**. В этой статье я использую две таблицы или модели: «*Категория*» и «*Продукт*«. Категория может содержать несколько продуктов. Сохраняем как файл **models.py**

models.py

```Python
from peewee import *

class BaseModel(Model):
```

```
    class Meta:
        database = dbhandle


class Category(BaseModel):
    id = PrimaryKeyField(null=False)
    name = CharField(max_length=100)

    created_at = DateTimeField(default=datetime.datetime.now())
    updated_at = DateTimeField(default=datetime.datetime.now())

    class Meta:
        db_table = "categories"
        order_by = ('created_at',)
```

Сначала я создал **BaseModel**. Это связанно с тем, что Peewee просит вас передать **dbhandle** в каждый класс Model. Чтобы избежать лишних движений, я просто создал базовый класс и расширил его. Ознакомиться со всеми типами столбцов в таблице можно тут: http://docs.peewee-orm.com/en/latest/peewee/models.html#field-types-table

Хорошо, наша BaseModel и Category созданы. Модель Category состоит из четырех полей:

1. **id**, который является полем автоматического прироста;
2. **name** содержит имя категории;
3. **updated_at** и **created_at** – поля **timestamp**, которые определяют настоящее время по умолчанию.

В классе **Meta** я передаю название таблицы в собственность **db_table**. Это не обязательно, если название таблицы и модели одинаковые. Собственность **order_by** указывает, какой столбец должен использоваться для **сортировки данных** во время извлечения. Вы можете переписать его, передав вид по полю на ваше усмотрение.

Перед тем как мы двинемся дальше, я хочу создать еще один файл под названием **operations.py**, в котором я буду использовать эти модели.

Python

```
import peewee
from models import *

if __name__ == '__main__':
    try:
        dbhandle.connect()
        Category.create_table()
    except peewee.InternalError as px:
        print(str(px))
```

После импорта, я подключаюсь к базе данных. Ошибка **peewee.OperationalError** ссылается на все ошибки, связанные с Peewee. К примеру, если вы введете неправильные учетные данные, вы получите следующее:

```
(1045, "Access denied for user 'root1'@'localhost' (using password: YES)")
```

Затем мы вызываем **Category.create_table()**, которые создает таблицу с указанной ранее собственностью. Если вы передаете **safe=True** в качестве параметра, то существующая таблица просто перепишется. Это может привести к проблемам в реальной ситуации.

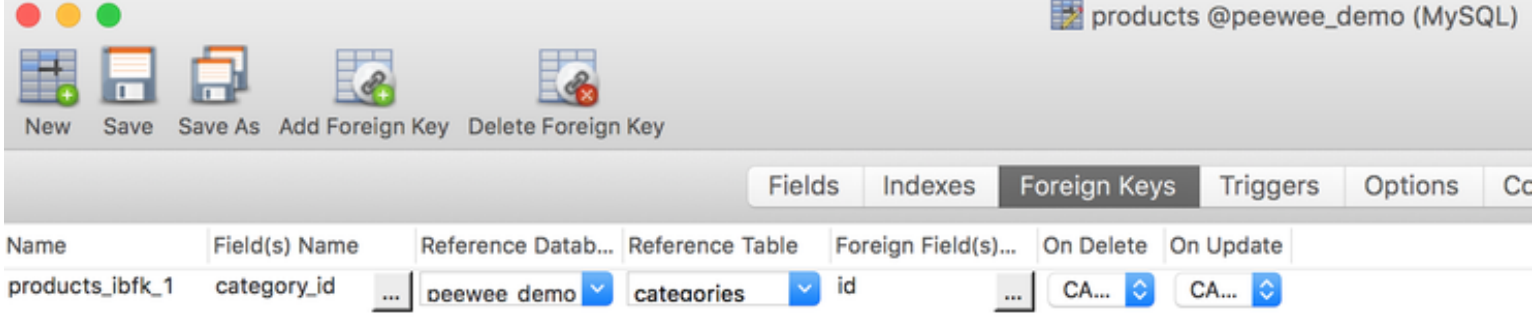
Далее, модель Product:

```python
class Product(BaseModel):
    id = PrimaryKeyField(null=False)
    name = CharField(max_length=100)
    price = FloatField(default=None)
    category = ForeignKeyField(Category, related_name='fk_cat_prod', to_field='id
                               on_update='cascade')
    created_at = DateTimeField(default=datetime.datetime.now())
    updated_at = DateTimeField(default=datetime.datetime.now())

    class Meta:
        db_table = "products"
        order_by = ('created_at',)
```

Она аналогична модели Cateogory. Разница только в **ForeignKeyField**, который указывает, как именно Product должен быть связан с **Category**. Обновление основы должно выглядеть следующим образом:

```python
if __name__ == '__main__':
    try:
        dbhandle.connect()
        Category.create_table()
    except peewee.InternalError as px:
        print(str(px))
    try:
        Product.create_table()
    except peewee.InternalError as px:
        print(str(px))
```

После запуска указанного выше кода, создается таблица модели **product**, а также отношение с таблицей **categories**. Вот скриншот моего клиента sql:



Вставка записи (INSERT)

Теперь мы можем перейти к добавлению данных сперва в **category**, а затем в таблицу **products**. Так как у нас есть рабочие модели, не так просто добавлять или обновлять записи.

Python

```python
import peewee
from models import *

def add_category(name):
    row = Category(
        name=name.lower().strip(),
    )
    row.save()

add_category('Books')
```

Я добавил функцию под названием **add_category**() с параметром и именем внутри. Объект Category создан, как и поля таблицы, которые являются переданной собственностью данного объекта класса. В нашем случае, это поле name.

The **row.save**() сохраняет данные из объекта в базу данных.

Круто, не так ли? Больше не нужно прописывать уродливые INSERT-ы.

Теперь добавим product.

Python

```python
import peewee
from models import *


def add_product(name, price, category_name):
    cat_exist = True
    try:
        category = Category.select().where(Category.name == category_name.strip()
    except DoesNotExist as de:
        cat_exist = False

    if cat_exist:
```

```
        row = Product(
            name=name.lower().strip(),
            price=price,
            category=category
        )
        row.save()
```

add_product берет **name**, **price** и **category_id** в качестве вводных данных. Сначала, я проверю, существует ли категория, если да – значит её объекты хранятся в базе. В ORM вы имеете дело с объектом, по этому вы передаете информацию о категории в качестве объекта, так как мы уже определили эту взаимосвязь ранее.

Далее, я буду создавать разделы в **main.py**:

Python
```
add_category('Books')
add_category('Electronic Appliances')
```

Теперь добавим продукты:

Python
```
# Добавление продуктов.
add_product('C++ Premier', 24.5, 'books')
add_product('Juicer', 224.25, 'Electronic Appliances')
```

Полный **main.py** можете видеть ниже:

main.py Python
```
import peewee
from models import *


def add_category(name):
    row = Category(
        name=name.lower().strip(),
    )
    row.save()


def add_product(name, price, category_name):
    cat_exist = True
    try:
        category = Category.select().where(Category.name == category_name.strip()
    except DoesNotExist as de:
        cat_exist = False

    if cat_exist:
        row = Product(
            name=name.lower().strip(),
            price=price,
            category=category
```

```
        )
        row.save()

if __name__ == '__main__':
    # Создаем разделы.
    add_category('Books')
    add_category('Electronic Appliances')

    # Добавляем продукты в разделы.
    add_product('C++ Premier', 24.5, 'books')
    add_product('Juicer', 224.25, 'Electronic Appliances')
```

Я передаю имя категории, как только её объект будет найден и передан объекту класса **Product**. Если вы хотите пойти по пути SQL, для начала вам нужно выполнить **SELECT**, чтобы получить существующий **category_id**, и затем назначить id добавляемому продукту.

Так как работать с **ORM** – значит иметь дело с объектами, мы храним объекты вместо скалярных значений. Кто-то может посчитать это слишком «*инженерным*» методом в нашем случае, но подумайте о случаях, когда вы понятия не имеете о том, какая база данных должна быть использована в будущем. Ваш код – это универсальный инструмент для работы с базами данных, так что если он работает с MySQL, или MSSQL, то он будет работать даже с MongoDb (гипотетически).

Выбор нескольких записей

Сначала выделяем категории:

Python

```
import peewee
from models import *


def find_all_categories():
    return Category.select()

def find_all_products():
    return Product.select()
```

Category.select() возвращает ВСЮ запись, которая будет отсортирована по столбцу **created_at**, что мы и указали в классе **Meta**.

Теперь выбираем все продукты:

Python

```
products = find_all_products()
product_data = []
for product in products:
    product_data.append({
        'title': product.name,
        'price': product.price,
        'category': product.category.name
    })

```

```
10
    print(product_data)
```

Здесь я перебираю продукты и добавляю запись в список **product_data**. Обратите внимание на доступ к категории продукта. Больше нет **SELECT** для получения ID, с последующим поиском названия. Простой цикл делает все за нас. При запуске, информация о продукте будет отображаться следующим образом:

```
Python
1 [
2     {'price': 24.5, 'title': 'c++ premier', 'category': 'books'},
3     {'price': 224.25, 'title': 'juicer', 'category': 'electronic appliances'}
4 ]
```

Выбор одной записи

Чтобы выбрать одну запись, вам нужно использовать **метод get**:

```
Python
1 def find_product(name):
2     return Product.get(Product.name == name.lower().strip())
```

Теперь это называется так:

```
Python
1 p = find_product('c++ premier')
2 print(p.category.name)
```

Название товара передано функции **find_product**, если запись существует – она вернет экземпляр **Product**. Здесь я вывожу категорию, связанную с этим продуктом.

Обновление записей в Peewee

Обновление записей – это так же просто, как и их создание. Вы получаете экземпляр объекта, после чего обновляете его.

```
Python
1 import peewee
2 from models import *
3
4 def update_category(id, new_name):
5     category = Category.get(Category.id == id)
6     category.name = new_name
7     category.save()
```

После этого, вызываем его как:

```
Python
1 update_category(2, 'Kindle Books')
```

Удаление записей в Peewee

Удаление записи не сильно отличается от обновления:

```python
import peewee
from models import *

def delete_category(name):
    category = Category.get(Category.name == name.lower().strip())
    category.delete_instance()
```

Вызываем данную функцию для удаления раздела:

```python
delete_category('Kindle Books')
```

Обратите внимание на то, что после удаления записи, вы также **удаляете связанные продукты**, так как они подключены к категории, и мы уже определили сценарий во время удаления раздела **ON_DELETE**, будут удаляться и товары из раздела.

Готовое приложение

Сохраняем данные криптовалют от биржи Binance

Допустим у нас уже есть MySQL база данных и существует таблица: **coins**. Нам нужно создать модель для этой существующей таблице и сохранить полученные json данные от Binance.

Структура базы данных:

```mysql
CREATE TABLE `coins` (
  `id` int(11) NOT NULL,
  `symbol` varchar(20) NOT NULL,
  `priceChange` float NOT NULL,
  `priceChangePercent` float NOT NULL,
  `weightedAvgPrice` float NOT NULL,
  `prevClosePrice` float NOT NULL,
  `lastPrice` float NOT NULL,
  `lastQty` float NOT NULL,
  `bidPrice` float NOT NULL,
  `askPrice` float NOT NULL,
  `openPrice` float NOT NULL,
  `highPrice` float NOT NULL,
  `lowPrice` float NOT NULL,
  `volume` float NOT NULL,
  `quoteVolume` float NOT NULL,
  `openTime` int(30) NOT NULL,
  `closeTime` int(30) NOT NULL,
```

```
  `firstId` int(10) NOT NULL,
  `lastId` int(10) NOT NULL,
  `count` int(10) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Данные от Binance мы получим по ссылке: https://api.binance.com/api/v1/ticker/24hr для HTTP запроса мы будет использовать библиотеку Requests на примере.

Что мы реализуем в нашем приложении:

Добавление новых монет;

Проверка если монета существует;

Обновления данных по цене если монета существует;

Сортируем монеты по объему за 24 часа;

Кол-во монет которые торгуются с BNB в паре (Binance Coin);

Выбираем случайные 5 монет

Исходный код

Python

```python
import requests
from peewee import MySQLDatabase, Model
from peewee import IntegerField, FloatField, CharField, PrimaryKeyField, Timesta
from peewee import InternalError

db = MySQLDatabase(
    'peewee', user='root', password='mangos',
    host='localhost'
)


class Coins(Model):
    id = PrimaryKeyField(null=False)
    symbol = CharField(unique=True)
    priceChange = FloatField()
    priceChangePercent = FloatField()
    weightedAvgPrice = FloatField()
    prevClosePrice = FloatField()
    lastPrice = FloatField()
    lastQty = FloatField()
    bidPrice = FloatField()
    askPrice = FloatField()
    openPrice = FloatField()
    highPrice = FloatField()
    lowPrice = FloatField()
    volume = FloatField()
    quoteVolume = FloatField()
    openTime = TimestampField()
    closeTime = TimestampField()
    firstId = IntegerField()
    lastId = IntegerField()
```

```
    count = IntegerField()

    class Meta:
        db_table = 'coins'
        database = db

# Создаем таблицу если не существует.
try:
    db.connect()
    Coins.create_table()
except InternalError as px:
    print(str(px))

# Получаем список криптовалют от Binance.
data = requests.get('https://api.binance.com/api/v1/ticker/24hr').json()

for coin in data:
    # Проверяем если валюта существует.
    exists = Coins.select().where(Coins.symbol == coin['symbol'])

    if bool(exists):
        print('Обновляем цены для:', coin['symbol'])
        # Запись существует.
        # Обновляем цены криптовалют.
        Coins.update(
            lastPrice=coin['lastPrice'],
            lastQty=coin['lastQty'],
            bidPrice=coin['bidPrice'],
            askPrice=coin['askPrice'],
        ).where(Coins.symbol == coin['symbol']).execute()
    else:
        print('Добавляем новую запись:', coin['symbol'])
        # Создаем новую запись.
        Coins.create(
            symbol=coin['symbol'],
            priceChange=coin['priceChange'],
            priceChangePercent=coin['priceChangePercent'],
            weightedAvgPrice=coin['weightedAvgPrice'],
            prevClosePrice=coin['prevClosePrice'],
            lastPrice=coin['lastPrice'],
            lastQty=coin['lastQty'],
            bidPrice=coin['bidPrice'],
            askPrice=coin['askPrice'],
            openPrice=coin['openPrice'],
            highPrice=coin['highPrice'],
            lowPrice=coin['lowPrice'],
            volume=coin['volume'],
            quoteVolume=coin['quoteVolume'],
            openTime=int(coin['openTime'] / 1000),
            closeTime=int(coin['closeTime'] / 1000),
            firstId=coin['firstId'],
            lastId=coin['lastId'],
            count=coin['count'],
        )

# Сортируем монеты по объему за 24 часа.
```

```python
sorted_coins = Coins.select().order_by(Coins.volume.desc())

print('Сортировка по объему за 24 часа')
for coin in sorted_coins:
    print(coin.symbol, coin.volume)

print('-' * 30)

# Кол-во монет которые торгуются с BNB в паре (Binance Coin).
bnb_count = Coins.select().where(Coins.symbol.endswith('BNB')).count()
print('Кол-во монет в паре с BNB:', bnb_count)

print('-' * 30)

# Выбираем случайные 5 монет.
random = Coins.select().order_by(fn.Rand()).limit(5)

print('Случайные 5 монет:')
for coin in random:
    print(coin.symbol)
```

В третьей строке вы можете видеть список типов столбцов из таблицы, такие как Integer, Float и Varchar (**IntegerField**, **FloatField**, **CharField**, **PrimaryKeyField**, **TimestampField**). Подробнее про тип полей можете узнать из документации: https://peewee.readthedocs.io/en/2.0.2/peewee/fields.html

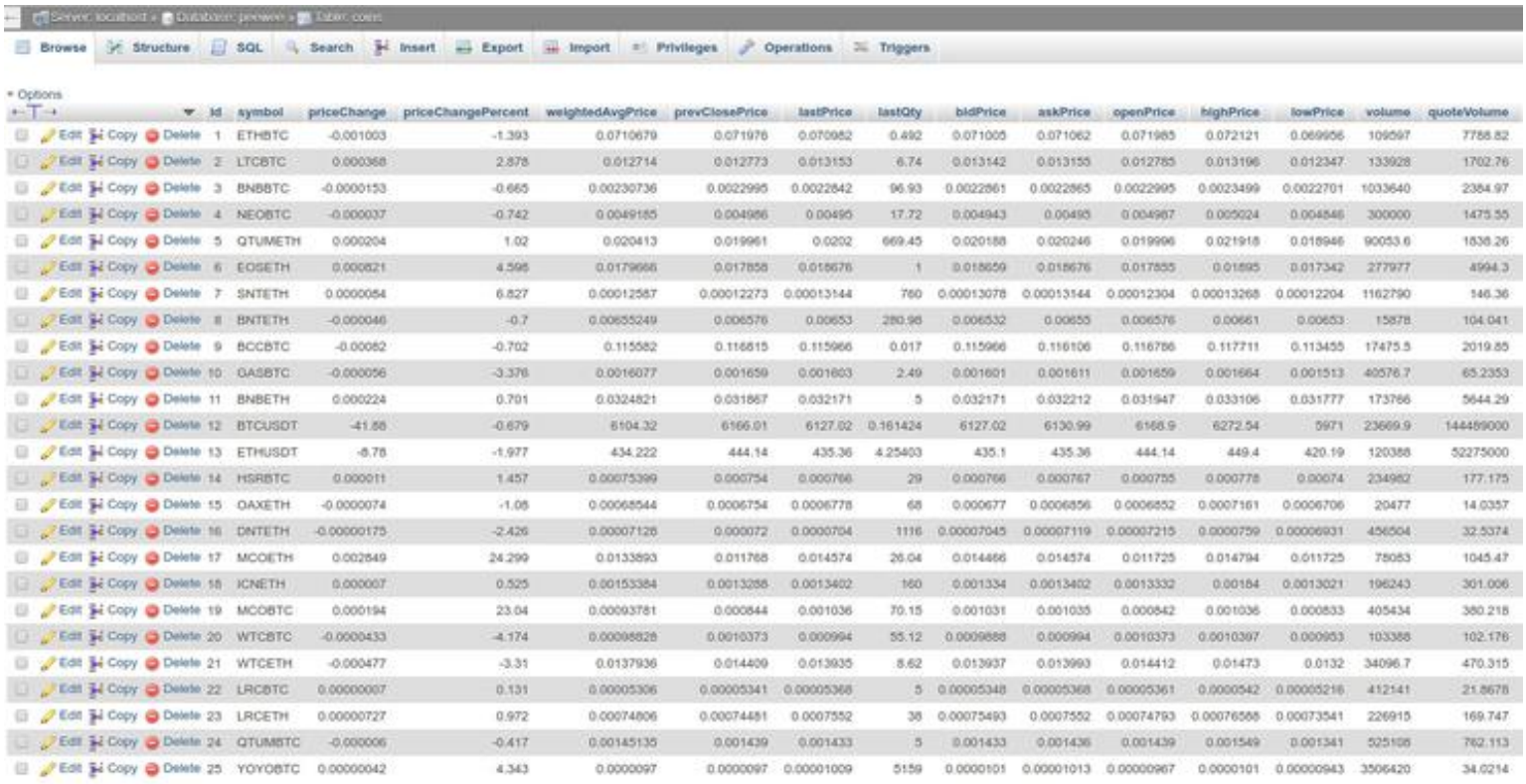
Больше примеров которые не были использованы: http://docs.peewee-orm.com/en/latest/peewee/querying.html

Результат который я получил:

Python

```
/usr/bin/python3.5 /home/database/peewee-test.py

Добавляем новую запись: ETHBTC
Добавляем новую запись: LTCBTC
....
Добавляем новую запись: KEYBTC
Добавляем новую запись: KEYETH

Сортировка по объему за 24 часа
NPXSBTC 1778630000.0
KEYBTC 782103000.0
....
BCCBNB 349.373
REPBNB 70.274
------------------------------
Кол-во монет в паре с BNB: 70
------------------------------
Случайные 5 монет:
THETABTC
POWRBNB
IOTABNB
AGIBTC
```

```
STEEMETH

Process finished with exit code 0
```



id	symbol	priceChange	priceChangePercent	weightedAvgPrice	prevClosePrice	lastPrice	lastQty	bidPrice	askPrice	openPrice	highPrice	lowPrice	volume	quoteVolume
1	ETHBTC	-0.001003	-1.393	0.0710679	0.071976	0.070982	0.492	0.071005	0.071062	0.071985	0.072121	0.069956	109597	7788.82
2	LTCBTC	0.000368	2.878	0.012714	0.012773	0.013153	6.74	0.013142	0.013155	0.012785	0.013196	0.012347	133928	1702.76
3	BNBBTC	-0.0000153	-0.665	0.00230736	0.0022995	0.0022842	96.93	0.0022861	0.0022865	0.0022995	0.0023499	0.0022701	1033640	2384.97
4	NEOBTC	-0.000037	-0.742	0.0049185	0.004986	0.00495	17.72	0.004943	0.00495	0.004987	0.005024	0.004846	300000	1475.55
5	QTUMETH	0.000204	1.02	0.020413	0.019961	0.0202	669.45	0.020188	0.020246	0.019996	0.021918	0.018946	90053.6	1838.26
6	EOSETH	0.000821	4.596	0.0179666	0.017858	0.018678	1	0.018659	0.018676	0.017855	0.01895	0.017342	277977	4994.3
7	SNTETH	0.0000054	6.827	0.00012587	0.00012273	0.00013144	760	0.00013078	0.00013144	0.00012304	0.00013268	0.00012204	1162790	146.36
8	BNTETH	-0.000046	-0.7	0.00655249	0.006576	0.00653	280.98	0.006532	0.00655	0.006576	0.00661	0.00653	15878	104.041
9	BCCBTC	-0.00062	-0.702	0.115582	0.116815	0.115966	0.017	0.115966	0.116106	0.116786	0.117711	0.113455	17475.5	2019.85
10	GASBTC	-0.000056	-3.376	0.0016077	0.001659	0.001603	2.49	0.001601	0.001611	0.001659	0.001664	0.001513	40576.7	65.2353
11	BNBETH	0.000224	0.701	0.0324821	0.031867	0.032171	5	0.032171	0.032212	0.031947	0.033106	0.031777	173766	5644.29
12	BTCUSDT	-41.66	-0.679	6104.32	6166.01	6127.02	0.161424	6127.02	6130.99	6168.9	6272.54	5971	23669.9	144459000
13	ETHUSDT	-6.78	-1.977	434.222	444.14	435.36	4.25403	435.1	435.36	444.14	449.4	420.19	120388	52275000
14	HSRBTC	0.000011	1.457	0.00075399	0.000754	0.000766	29	0.000766	0.000767	0.000755	0.000778	0.00074	234982	177.175
15	OAXETH	-0.0000074	-1.08	0.00068544	0.0006754	0.0006778	68	0.000677	0.0006856	0.0006852	0.0007161	0.0006706	20477	14.0357
16	DNTETH	-0.00000175	-2.426	0.00007126	0.000072	0.0000704	1116	0.00007045	0.00007119	0.00007215	0.0000759	0.00006931	456504	32.5374
17	MCOETH	0.002849	24.299	0.0133893	0.011768	0.014574	26.04	0.014466	0.014574	0.011725	0.014794	0.011725	78083	1045.47
18	ICNETH	0.000007	0.525	0.00153384	0.0013288	0.0013402	160	0.001334	0.0013402	0.0013332	0.00184	0.0013021	196243	301.006
19	MCOBTC	0.000194	23.04	0.00093781	0.000844	0.001036	70.15	0.001031	0.001035	0.000842	0.001036	0.000833	405434	380.218
20	WTCBTC	-0.0000433	-4.174	0.00098828	0.0010373	0.000994	55.12	0.0009888	0.000994	0.0010373	0.0010397	0.000953	103388	102.176
21	WTCETH	-0.000477	-3.31	0.0137936	0.014409	0.013935	8.62	0.013937	0.013993	0.014412	0.01473	0.0132	34096.7	470.315
22	LRCBTC	0.00000007	0.131	0.00005306	0.00005341	0.00005368	5	0.00005348	0.00005368	0.00005361	0.0000542	0.00005216	412141	21.8678
23	LRCETH	0.00000727	0.972	0.00074806	0.00074481	0.0007552	38	0.00075493	0.0007552	0.00074793	0.00076568	0.00073541	226915	169.747
24	QTUMBTC	-0.000006	-0.417	0.00145135	0.001439	0.001433	5	0.001433	0.001436	0.001439	0.001549	0.001341	525108	762.113
25	YOYOBTC	0.00000042	4.343	0.0000097	0.0000097	0.00001009	5159	0.0000101	0.00001013	0.00000967	0.0000101	0.00000943	3506420	34.0214

Меняем MySQL на SQLite

Используя пример из кода выше, мы изменим базу данных из MySQL на SQLite не трогая код (кроме самого подключения). Нам нужно обновить немного наш файл. С самых первых строк было:

```python
import requests
from peewee import MySQLDatabase, Model
from peewee import IntegerField, FloatField, CharField, PrimaryKeyField, Timestamp
from peewee import InternalError

db = MySQLDatabase(
    'peewee', user='root', password='mangos',
    host='localhost'
)
```

Теперь у нас:

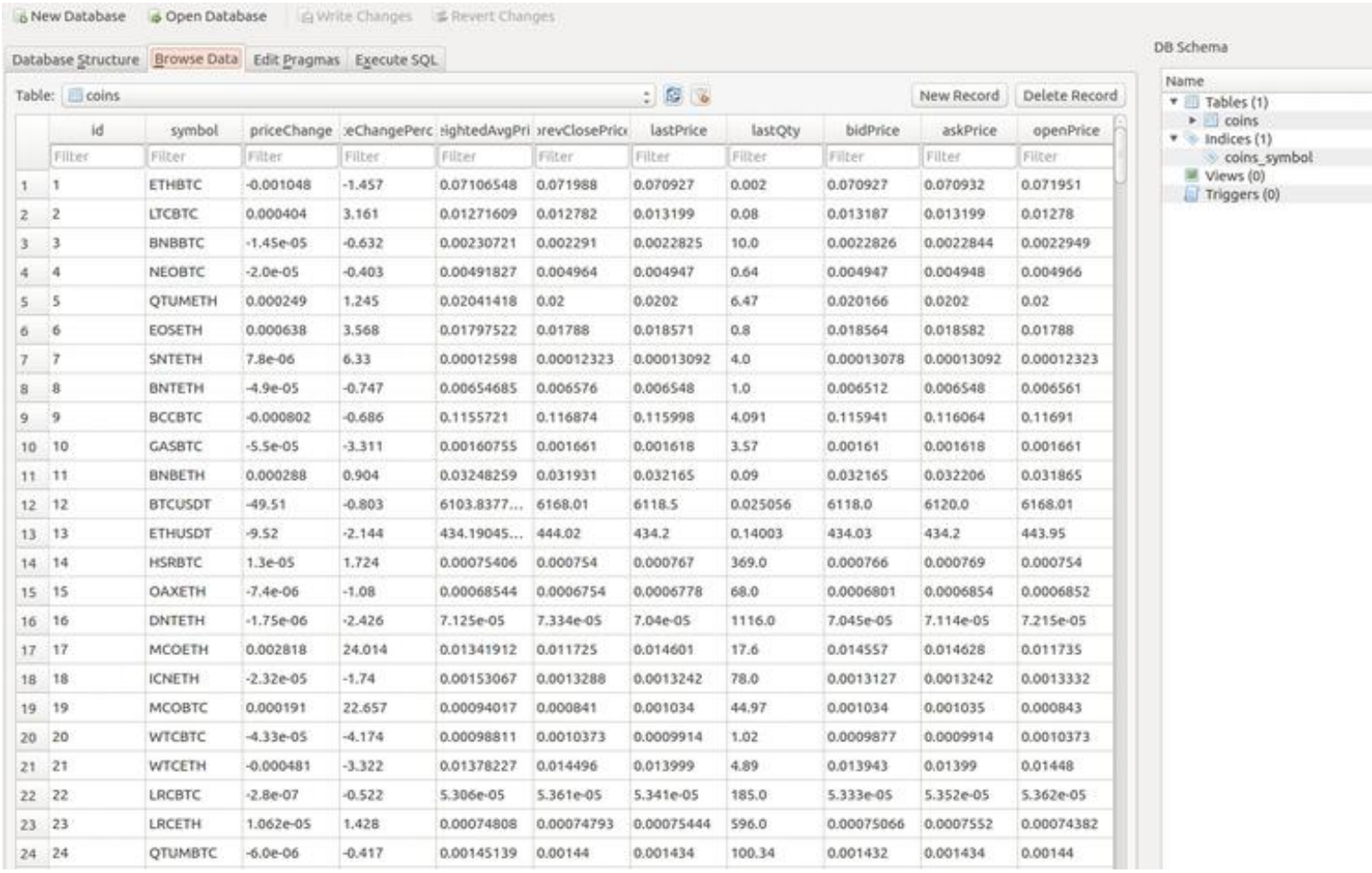
```python
import requests
from peewee import SqliteDatabase, Model
from peewee import IntegerField, FloatField, CharField, PrimaryKeyField, Timestamp
from peewee import InternalError

db = SqliteDatabase('binance-coins.db')
```

После запуска отредактированного кода мы получим ошибку:

```
peewee.OperationalError: no such function: Rand
```

Ошибка в примере «Выбираем случайные 5 монет», в коде мы используем **fn.Rand**() и он работает для MySQL, но для SQLite и PostgreSQL он работать не будет, нужно использовать **fn.Random**().
Подробнее: http://docs.peewee-orm.com/en/latest/peewee/querying.html#getting-random-records

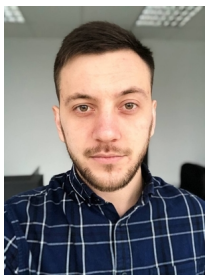
После запуска у нас появился файл базы данных **binance-coins.db**



	id	symbol	priceChange	eChangePerc	eightedAvgPri	revClosePric	lastPrice	lastQty	bidPrice	askPrice	openPrice
1	1	ETHBTC	-0.001048	-1.457	0.07106548	0.071988	0.070927	0.002	0.070927	0.070932	0.071951
2	2	LTCBTC	0.000404	3.161	0.01271609	0.012782	0.013199	0.08	0.013187	0.013199	0.01278
3	3	BNBBTC	-1.45e-05	-0.632	0.00230721	0.002291	0.0022825	10.0	0.0022826	0.0022844	0.0022949
4	4	NEOBTC	-2.0e-05	-0.403	0.00491827	0.004964	0.004947	0.64	0.004947	0.004948	0.004966
5	5	QTUMETH	0.000249	1.245	0.02041418	0.02	0.0202	6.47	0.020166	0.0202	0.02
6	6	EOSETH	0.000638	3.568	0.01797522	0.01788	0.018571	0.8	0.018564	0.018582	0.01788
7	7	SNTETH	7.8e-06	6.33	0.00012598	0.00012323	0.00013092	4.0	0.00013078	0.00013092	0.00012323
8	8	BNTETH	-4.9e-05	-0.747	0.00654685	0.006576	0.006548	1.0	0.006512	0.006548	0.006561
9	9	BCCBTC	-0.000802	-0.686	0.1155721	0.116874	0.115998	4.091	0.115941	0.116064	0.11691
10	10	GASBTC	-5.5e-05	-3.311	0.00160755	0.001661	0.001618	3.57	0.00161	0.001618	0.001661
11	11	BNBETH	0.000288	0.904	0.03248259	0.031931	0.032165	0.09	0.032165	0.032206	0.031865
12	12	BTCUSDT	-49.51	-0.803	6103.8377...	6168.01	6118.5	0.025056	6118.0	6120.0	6168.01
13	13	ETHUSDT	-9.52	-2.144	434.19045...	444.02	434.2	0.14003	434.03	434.2	443.95
14	14	HSRBTC	1.3e-05	1.724	0.00075406	0.000754	0.000767	369.0	0.000766	0.000769	0.000754
15	15	OAXETH	-7.4e-06	-1.08	0.00068544	0.0006754	0.0006778	68.0	0.0006801	0.0006854	0.0006852
16	16	DNTETH	-1.75e-06	-2.426	7.125e-05	7.334e-05	7.04e-05	1116.0	7.045e-05	7.114e-05	7.215e-05
17	17	MCOETH	0.002818	24.014	0.01341912	0.011725	0.014601	17.6	0.014557	0.014628	0.011735
18	18	ICNETH	-2.32e-05	-1.74	0.00153067	0.0013288	0.0013242	78.0	0.0013127	0.0013242	0.0013332
19	19	MCOBTC	0.000191	22.657	0.00094017	0.000841	0.001034	44.97	0.001034	0.001035	0.000843
20	20	WTCBTC	-4.33e-05	-4.174	0.00098811	0.0010373	0.0009914	1.02	0.0009877	0.0009914	0.0010373
21	21	WTCETH	-0.000481	-3.322	0.01378227	0.014496	0.013999	4.89	0.013943	0.01399	0.01448
22	22	LRCBTC	-2.8e-07	-0.522	5.306e-05	5.361e-05	5.341e-05	185.0	5.333e-05	5.352e-05	5.362e-05
23	23	LRCETH	1.062e-05	1.428	0.00074808	0.00074793	0.00075444	596.0	0.00075066	0.0007552	0.00074382
24	24	QTUMBTC	-6.0e-06	-0.417	0.00145139	0.00144	0.001434	100.34	0.001432	0.001434	0.00144

Вывод

Отлично, вы освоили **основы Peewee** и то, как вы можете использовать эту маленькую и эффективную ORM в ваших следующих проектах, связанных с Pyhton. Если у вас есть дополнительные вопросы – вы можете ознакомиться с официальной документацией и найти в ней ответы.



Vasile Buldumac

Являюсь администратором нескольких порталов по обучению языков программирования Python, Golang и Kotlin. В составе небольшой команды единомышленников, мы занимаемся популяризацией языков программирования на русскоязычную аудиторию. Большая часть статей была адаптирована нами на русский язык и распространяется бесплатно.

E-mail: vasile.buldumac@ati.utm.md

Образование
Universitatea Tehnică a Moldovei (*utm.md*)

2014 — 2018 Технический Университет Молдовы, ИТ-Инженер. Тема дипломной работы «*Автоматизация покупки и продажи криптовалюты используя технический анализ*»

2018 — 2020 Технический Университет Молдовы, Магистр, Магистерская диссертация «*Идентификация человека в киберпространстве по фотографии лица*»