УДК 004.624

Использование инструмента Pony ORM для создания и обслуживания баз данных

Козич Полина Александровна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Научный руководитель:
Глаголев Владимир Александрович
Приамурский государственный университет имени Шолом-Алейхема
к.г.н., доцент кафедры информационных систем, математики и правовой информатики

Аннотация
В данной статье рассказано, что из себя, представляет инструмент PonyORM, описаны его достоинства и ключевые особенности. Рассмотрены режимы работы и основные приемы применения при работе с базами данных.
Ключевые слова: Python, PonyORM

Using the tool Pony ORM to create and maintain databases

Kozich Polina Alexandrovna
Sholom-Aleichem Priamursky State University
Student

Scientific adviser:
Glagolev Vladimir Alexandrovich
Sholom-Aleichem Priamursky State University
candidate of geographical sciences, Associate Professor of the Department of Information System, Mathematics and law informatics

Abstract
This article will explain what PonyORM is all about, describing its merits and key features. Considered modes of operation and the basic techniques of application when working with databases.
Keywords: Python, PonyORM

Pony ORM — это пакет реляционных объектов для Python. Он позволяет запрашивать данные из базы данных с помощью конструкций языка Python. Pony способствует быстрой разработке приложений.

Pony — это легкая в освоении и работе библиотека, что делает работу с ней более продуктивной и помогает экономить ресурсы. Pony достигает этой простоты использования с помощью, следующего:

- Компактные определения сущностей
- Краткий язык запросов
- Возможность интерактивной работы с Pony в интерпретаторе Python
- Комплексные сообщения об ошибках, показывающие конкретную часть, где произошла ошибка в запросе
- Отображение сгенерированного SQL в читаемом формате с отступами

Все это помогает разработчику сосредоточиться на реализации бизнес-логики приложения вместо того, чтобы отвлекаться на SQL запросы, пытаясь понять, как лучше получить данные из базы данных.

Цель исследования – дать базовое понимание работы с системой PonyORM и понимание преимуществ её применения.

Ранее этим вопросом интересовались Е. В. Ковалевская, Г. А. Черменнов развивали тему « Разработка кода для доступа к данным с применением объектно-реляционного отображения» [2] в которой обсуждается вопрос разработки кода для доступа к данным с применением объектно-реляционного отображения. Под объектно-реляционным отображением понимается метод преобразования данных, позволяющий из набора таблиц из реляционной базы данных получить объектную модель, а также осуществить обратный переход - от объектной структуры перейти к записям и столбцам в таблицах базы данных. Ю.Г. Платонов, Е.В. Артамонова с темой «Возможности технологии object relational mapping для нереляционных баз данных» [3], исследование посвящено вопросу разработки новых способов использования специализированных БД при работе с большими данными, и, в том числе, использования их в роли СУБД. В качестве тестового полигона использовалась разработана специализированная система Polar (оригинальная разработка САПР и А СБИС ИСИ СО РАН). Р.В. Ворожейкин опубликовал статью «Объектно-реляционное отображение базы данных на примере Django» [4] рассмотрел преимущества и недостатки использования объектно-реляционного отображения базы данных на примере фреймворка Django. Описывает понятие объектно-реляционного отображения и его достоинств, реализованном во фреймворке Django. Автор приводит список встроенных полей в ORM Django. Также приводит пример описания простой модели при помощи ORM Django и показывает как эта модель будет преобразована в SQL с синтаксисом PostgreSQL.

Поскольку этот проект не входит в состав Python, необходимо скачать и установить его. Установить его можно благодаря следующей команде:

```
pip install pony
```

Работа с системой PonyORM будет представлена на примере создания базы данных для хранения певцов и их музыки.

Создание базы данных

Для начала создадим две таблицы: Art и Alb.

```
import datetime
import pony.orm as pony

database = pony.Database('sqlite',
                         'music.sqlite',
                         create_db=True)
 class Art(database.Entity):
    name_art = pony.Required(unicode)
    albs = pony.Set('Alb')

class Alb(database.Entity):
    Art = pony.Required(Art)
    title_alb = pony.Required(unicode)
    release_date = pony.Required(datetime.date)
    publisher = pony.Required(unicode)
    media_type = pony.Required(unicode)

pony.sql_debug(True)

database.generate_mapping(create_tables=True)
```

Pony ORM создаст первичный ключ автоматически, если его не указать явно. Чтобы создать внешний ключ, все, что нужно сделать, это передать класс модели в другую таблицу, как мы делали в классе Alb. Каждое обязательное поле принимает тип Python. Большинство полей содержат юникод, одно из которых является объектом времени. Затем включаем режим отладки, который выводит SQL, который Pony генерирует при создании таблиц в последнем операторе. Важным моментом является то, что если база уже существует, то PonyORM не будет пересоздавать новую базы. Pony проверит, существуют ли таблицы, прежде чем создавать их.

После выполнения кода должно быть напечатано следующее.

```
GET CONNECTION FROM THE LOCAL POOL
PRAGMA foreign_keys = false
BEGIN IMMEDIATE TRANSACTION
CREATE TABLE 'Art' (
  'id' INTEGER PRIMARY KEY AUTOINCREMENT,
  'name_art' TEXT NOT NULL
)

CREATE TABLE 'Alb' (
  'id' INTEGER PRIMARY KEY AUTOINCREMENT,
  'art' INTEGER NOT NULL REFERENCES 'Art' ('id'),
  'title_alb' TEXT NOT NULL,
  'release_date' DATE NOT NULL,
  'publisher' TEXT NOT NULL,
  'media_type' TEXT NOT NULL
)

CREATE INDEX 'idx_alb__art' ON 'Alb' ('art')

SELECT 'Alb'.'id', 'Alb'.'art', 'Alb'.'title_alb', 'Alb'.'release_date',
'Alb'.'publisher', 'Alb'.'media_type'
FROM 'Alb' 'Alb'
WHERE 0 = 1
```

```
SELECT 'Art'.'id', 'Art'.'name_art'
FROM 'Art' 'Art'
WHERE 0 = 1

COMMIT
PRAGMA foreign_keys = true
CLOSE CONNECTION
```

Все, база данных создана.

Как вставить / добавить данные в таблицы

Pony делает добавление данных в таблицы достаточно легким. Пример добавления описан ниже.

```
import datetime
import pony.orm as pony
from models import Alb, Art

@pony.db_session
def add_data():
    new_art = Art(name_art='Newsboys')
    bands = ['MXPX', 'Kutless', 'Thousand Foot Krutch']
    for band in bands:
        art = Art(name_art=band)

    alb = Alb(art=new_art,
                  title_alb='Read All About It',
                  release_date=datetime.date(1988,12,01),
                  publisher='Refuge',
                  media_type='CD')
    albs = [{'art': new_art,
               'title_alb': 'Hell is for Wimps',
               'release_date': datetime.date(1990,07,31),
               'publisher': 'Sparrow',
               'media_type': 'CD'
               },
              {'art': new_art,
               'title_alb': 'Love Liberty Disco',
               'release_date': datetime.date(1999,11,16),
               'publisher': 'Sparrow',
               'media_type': 'CD'
              },
              {'art': new_art,
               'title_alb': 'Thrive',
               'release_date': datetime.date(2002,03,26),
               'publisher': 'Sparrow',
               'media_type': 'CD'}
              ]

    for alb in albs:
        a = Alb(**alb)

if __name__ == '__main__':
    add_data()

    with pony.db_session:
        a = Art(name_art='Skillet')
```

При каждом заполнении базы данных нужно использовать декаратор db_session, он проверяет открытие соединения с базой данных, фиксации данных и закрытии соединения.

Использование базовых запросов для изменения записей

На примере ниже описан способ получения данных из базы данных и редактировании некоторых из них.

```
import pony.orm as pony
from models import Art, Alb

with pony.db_session:
    band = Art.get(name_art='Newsboys')
    print(band.name_art)

    for record in band.albs:
        print(record.title_alb)

    band_name_art = Art.get(name_art='Kutless')
    band_name_art.name_art = 'Beach Boys'
```

Делаем запрос, чтобы получить объект Art из базы данных и напечатать его имя. Затем перебираем альбомы исполнителя, которые также содержатся в возвращаемом объекте. Наконец, меняем одно из имен артиста.

Ниже представлен код для получения данных с помощью генератора:

```
result = pony.select(i.name_art for i in Art)
result.show()
```

Если запустить этот код, можно увидеть что-то вроде следующего:

(i.name_art

Newsboys
MXPX
Beach Boys
Thousand Foot Krutch)

Как удалить записи в Pony ORM

Удаление записей с Pony также довольно легко. Для примера удалим одну строку из базы данных:

```
import pony.orm as pony

from models import Art

with pony.db_session:
    band = Art.get(name_art='MXPX')
    band.delete()
```

Для удаления записи используется метод delete объекта band.

Вывод

Теперь мы знакомы с основами использования пакета Pony ORM. Использование данного пакета существенно ускоряет разработку и снимает дополнительное расходы на изучение дополнительных языков SQL, потому как теперь не нужно задумываться о том как связать свое приложение и базу данных, а достаточно описать несколько инструкций на языке Python для создания базы данных и последующего её использования.

Библиографический список

1. Object-Relational Mapper // PonyORM URL: https://ponyorm.org/ (дата обращения: 05.11.2019).
2. Ковалевская Е.В., Черменнов Г.А. Разработка кода для доступа к данным с применением объектно-реляционного отображения // Казанский социально-гуманитарный вестник. 2013. № 1 (7). С. 15-17.
3. Платонов Ю.Г., Артамонова Е.В. Возможности технологии object relational mapping для нереляционных баз данных // Современные тенденции развития науки и технологий. 2017. № 3-4. С. 104-108.
4. Ворожейкин Р.В. объектно-реляционное отображение базы данных на примере Django // Информационно-компьютерные технологии в экономике, образовании и социальной сфере. 2016. № 2 (12). С. 67-73.