УДК 681.3.06

М. Я. Афанасьев, А. Н. Филиппов

СОЗДАНИЕ ДИНАМИЧЕСКИХ МОДЕЛЕЙ БАЗ ДАННЫХ ТЕХНОЛОГИЧЕСКОГО НАЗНАЧЕНИЯ НА ЯЗЫКЕ PYTHON

Рассмотрен метод создания моделей баз данных на языке сверхвысокого уровня Python. Проведен сравнительный анализ метода с классическими.

Ключевые слова: *базы данных, технология приборостроения.*

С момента своего создания системы автоматизированного проектирования технологических процессов (САПР ТП) тесно связаны с базами данных (БД). Базы данных технологического назначения позволяют структурировать и организовывать данные, описывающие характеристики различных объектов, необходимых для проектирования и управления технологическими процессами (ТП). Это очень удобно, но до тех пор пока не появляется необходимость смены формата баз данных, изменения структуры данных или перехода к новой САПР ТП. Поэтому проблема совместимости между различными форматами БД и конвертации из одного формата в другой является основной при проектировании систем управления базами данных технологического назначения (СУБД ТН).

Классический способ работы с БД предполагает подключение к серверу базы данных, получение от него некоторых данных (с помощью языка запросов), форматирование и отображение полученных данных. Например, для обращения к базе данных режущего инструмента (в формате mySQL) и вывода наименований всех резцов можно создать следующую функцию на языке Python:

```
# импортирование модуля доступа к БД mySQL
import MySQLdb

def tool_bits_list():
    # подключение к серверу БД
    db = MySQLdb.connect(
        user = 'ivanov',
        db = 'ctoolsdb',
        passwd = 'secret',
        host = 'localhost'
    )
```

```
    # создание курсора
    cursor = db.cursor()
    # исполнение SQL-запроса
    cursor.execute( 'SELECT name FROM tool_bits ORDER BY name' )
    # получение массива имен
    names = [ row[ 0 ] for row in cursor.fetchall() ]
    # закрытие соединения с сервером БД
    db.close()
    return names.
```

Данный подход прекрасно работает, но очевидны следующие проблемы.

1. В функции жестко определяются параметры соединения с базой данных, в идеале эти параметры должны храниться в конфигурации системы.

2. Необходимо разрабатывать дополнительный код для создания соединения, курсора, выполнения оператора и закрытия соединения.

3. Данный подход привязывает систему к БД MySQL. В случае необходимости перехода от СУБД MySQL к СУБД PostgreSQL или, например, Oracle потребуется использовать другой драйвер базы данных, изменять параметры соединения и, в зависимости от природы SQL-операторов, возможно, переписать SQL-запросы.

Для решения вышеописанных проблем предлагается использовать так называемые модели БД. Модель представляет собой описание данных, которые хранятся в базе данных, выполненное в виде кода на языке Python. Такая форма представления данных — эквивалент SQL-операторов CREATE TABLE — она описана на языке Python и включает в себя не только определение столбцов в базе данных. Система использует модель для фонового выполнения SQL-запроса и возвращает структуры Python (список, ассоциативный массив или кортеж) с данными, представляющими собой записи в таблицах базы данных. Модели также могут использоваться для представления высокоуровневых концепций, которые язык SQL вряд ли сможет обработать [2, 3].

Может показаться, что введение дополнительного уровня представления данных бессмысленно, но для этого есть несколько причин:

— использование модели позволяет работать с любыми форматами БД без дополнительных перенастроек, а также параллельно с несколькими различными БД в одном проекте;

— нет необходимости в написании сложных SQL-запросов;

— когда модели баз данных хранятся в виде сценария, а не в базе данных, упрощается управление версиями этих моделей. Это помогает сохранить историю всех изменений моделей;

— существует неполная совместимость языка SQL с разными платформами баз данных. При распространении САПР ТП более практично передавать модуль на языке Python, который описывает формат данных вместо отдельных наборов операторов CREATE TABLE для MySQL, PostgreSQL, SQLite, Oracle и др.

Таким образом, модель базы данных режущего инструмента может быть представлена следующим образом:

```
from db import models

# Таблица резцов
class ToolBit( models.Model ):
    name = models.CharField( max_length=50 )
    gost = models.CharField( max_length=30 )
    material = models.CharField( max_length=20 )
    type = models.IntField( max_length=3 )
```

```
# Таблица фрез
class MillingCutters( models.Model ):
    name = models.CharField( max_length=50 )
    gost = models.CharField( max_length=30 )
    material = models.CharField( max_length=20 )
    type = models.IntField( max_length=2 )
    diameter = models.FloatField( max_digits=5, decimal_places=2)
    flutes = models.IntField( max_length=2 )
    coating = models.CharField( max_length=30 )
    helix_angle = models.FloatField( max_digits=5, decimal_places=3 )

# Таблица сверл
class DrillBits( models.Model ):
    name = models.CharField( max_length=50 )
    gost = models.CharField( max_length=30 )
    material = models.CharField( max_length=20 )
    type = models.IntField( max_length=2 )
    length = models.FloatField( max_digits=5, decimal_places=2 )
    diameter = models.FloatField( max_digits=5, decimal_places=2 ).
```

Можно видеть, что все классы описанной выше модели наследуются от базового класса models. Методы CharField, IntField и FloatField определяют тип поля таблицы (строковый, целый и вещественный), а параметры этих методов задают формат (маску) поля. Дополнительно могут быть введены и другие типы полей (например, поля даты, времени, отношений „один-ко-многим“ и „многие-ко-многим“, поля для представления внешнего ключа и т.д.)

После создания модели пользователю автоматически предоставляется высокоуровневый API (Application Programming Interface) для работы с ней [4, 5]. Поясним это на конкретном примере:

```
from db.models import ToolBit
#Создаем новую запись с помощью конструктора
record = ToolBit(
    name = 'TB1'
    gost = 'GOST'
    material = 'steel'
    type = 10
)
#Сохраняем запись
record.save().
```

Подобная запись гораздо удобнее и нагляднее традиционного SQL-запроса. Также API предоставляет возможность простого доступа к данным, например, для изменения поля name достаточно написать следующее:

```
record.name = "TB2"
record.save(),
```

система самостоятельно определит ключ изменяемой записи и заменит значение только одного поля. Чтобы проделать то же с помощью SQL-запроса, нам было бы необходимо сначала каким-то образом определить ключ записи, а затем создать следующий запрос:

```
UPDATE db SET
    name = 'TB1'
    gost = 'GOST'
    material = 'steel'
    type = 10
WHERE index = 10.
```

Фильтрация данных осуществляется с помощью метода `filter()`, например:

```
#Выбрать все резцы десятого типа
ToolBit.objects.filter( type = 10 )
#Тоже самое, но используется LIKE вместо операции равенства
ToolBit.objects.filter( type__contains = 10 )
#Выбрать все стальные резцы десятого типа (операция AND)
ToolBit.objects.filter( type = 10, material = 'steel' ).
```

Доступны и другие типы выборок, включая `icontains` (не зависящий от регистра LIKE), `startswith` (записи начинающиеся с) и `endswith` (записи, заканчивающиеся на), `range` (SQL-оператор BETWEEN) и др.

Сортировка осуществляется с помощью метода `order_by()`:

```
#Прямая сортировка по полю name
ToolBit.objects.order_by( "name" )
#Обратная сортировка по полю name
ToolBit.objects.order_by( "-name" )
#Прямая сортировка по полям name и type
ToolBit.objects.order_by( "name", "type" )
#Сортировка по умолчанию (поле должно быть задано в метаданных моде-
ли)
ToolBit.objects.order_by().
```

Можно сортировать и фильтровать данные одновременно:

```
ToolBit.objects.filter( type = 10 ).order_by( "name" ).
```

Дополнительно можно делать частичные выборки, удалять записи и т.д. Хорошо продуманная модель позволяет забыть о громоздких и неудобных в проектировании SQL-запросах, но несмотря на все вышеописанные достоинства данный подход имеет один недостаток: при изменении модели структура БД не меняется. Эта проблема носит чисто технический характер и со временем, несомненно, будет устранена.

СПИСОК ЛИТЕРАТУРЫ

1. *Филиппов А. Н.* Разработка и исследование методов экспертных систем в САПР ТП механической обработки. Дис. … канд. техн. наук. Л.: ЛИТМО, 1991. 148 с.

2. *Бизли Д. М.* Язык программирования Python. Справочник. К.: ДиаСофт, 2000. 336 с.

3. *Лутц М.* Программирование на Python. СПб: Символ-Плюс, 2002. 1136 с.

4. *Сузи Р. А.* Python. Наиболее полное руководство. СПб: БХВ-Петербург, 2002. 768 с.

5. *Сузи Р. А.* Язык программирования Python: Учеб. пособие. М.: ИНТУИТ, БИНОМ. Лаборатория знаний, 2006. 328 с.

Сведения об авторах

Максим Яковлевич Афанасьев — аспирант; Санкт-Петербургский государственный университет информационных технологий, механики и оптики, кафедра технологии приборостроения; E-mail: ichiro.kodachi@gmail.com

Александр Николаевич Филиппов — канд. техн. наук, доцент; Санкт-Петербургский государственный университет информационных технологий, механики и оптики, кафедра технологии приборостроения; E-mail: filippov_an@rambler.ru

Рекомендована кафедрой технологии приборостроения

Поступила в редакцию 14.12.09 г.