Objektno-relaciono mapiranje

Most između dva sveta

Objektno-relaciono mapiranje: Most između dva sveta

Objektno-relaciono mapiranje (ORM) predstavlja jednu od najznačajnijih tehnoloških inovacija u razvoju softvera, omogućavajući programerima da rade sa bazama podataka koristeći objektno-usmerene pojmove umesto neposrednog pisanja SQL upita. Ova tehnologija iz osnova menja način na koji razmišljamo o odnosu između programskog koda i podataka.

Suština ORM-a: Prevazilaženje nesaglasnosti između modela

ORM rešava problem koji se u softverskom inženjerstvu naziva "nesaglasnost impedanse" - neusklađenost između objektnog modela programskog jezika i relacionog modela baze podataka. Relacione baze podataka organizuju podatke u tabele sa redovima i stubcima, dok objektno-usmereni jezici koriste klase, objekte i nasleđivanje.

python

```
# Bez ORM-a - neposredan SQL
cursor.execute("SELECT * FROM korisnici WHERE godine > %s",
(18,))
korisnici = cursor.fetchall()
for korisnik in korisnici:
    print(f"Korisnik: {korisnik[1]}, Godine: {korisnik[2]}")

# Sa ORM-om - objektno-usmeren pristup
korisnici = Korisnik.objects.filter(godine__gt=18)
for korisnik in korisnici:
    print(f"Korisnik:         {korisnik.ime},         Godine:
{korisnik.godine}")
```

Prednosti ORM-a: Više od obične jednostavnosti

1. Apstrakcija složenosti

ORM omogućava programerima da se usredsrede na poslovnu logiku umesto na tehničke pojedinosti baze podataka. Ovo je

posebno važno u velikim timovima gde različiti članovi imaju različite nivoe iskustva sa bazama podataka.

2. Prenosljivost između baza podataka

Jedna od najznačajnijih prednosti ORM-a je mogućnost prelaska između različitih sistema baza podataka bez menjanja programskog koda:

python

```
# Isti kod radi sa PostgreSQL, MySQL, SQLite...
class Proizvod(Model):
    naziv = CharField(max_length=100)
    cena = DecimalField(max_digits=10, decimal_places=2)
    kreiran = DateTimeField(auto_now_add=True)
```

3. Bezbednost i zaštita od SQL napada

ORM automatski parametrizuje upite, što značajno smanjuje opasnost od SQL napada ubrizgavanjem koda.

Mane i ograničenja: Kritički pogled

1. Dodatno opterećenje performansi

ORM često stvara neoptimalne SQL upite. Problem "N+1 upita" je klasičan primer:

python

```
# Loš pristup - stvara N+1 upita
knjige = Knjiga.objects.all()
for knjiga in knjige:
    print(knjiga.autor.ime)   # Svaki pristup autoru stvara
novi upit

# Optimizovan pristup
knjige = Knjiga.objects.select_related('autor')
for knjiga in knjige:
    print(knjiga.autor.ime)  # Jedan upit sa spajanjem tabela
```

2. Složenost u naprednim slučajevima

Za napredne SQL operacije, ORM može biti ograničavajući, zahtevajući kombinovanje sa "sirovim" SQL upitima.

Nekonvencionalni pristup: Stvaralačka upotreba ORM-a

1. ORM kao sloj za API

Umesto korišćenja ORM-a samo za bazu podataka, možete ga proširiti za rad sa različitim izvorima podataka:

python

```
class APIModel(Model):
    class Meta:
        managed = False  # Django neće kreirati tabelu

    def save(self):
        # Umesto u bazu, šalje podatke u API
        odgovor                                                    =
requests.post('https://api.primer.com/podaci',
                                  data=self.u_recnik())
        return odgovor.json()
```

2. Vremenske serije kroz ORM

Stvaralačko korišćenje ORM-a za analizu vremenskih serija:

python

```
class VremenskeSerije(QuerySet):
    def vremenska_grupa(self, interval='1 sat'):
        return self.extra(
            select={'vremenska_grupa':
                    f"DATE_TRUNC('{interval}', kreiran)"}
        ).values('vremenska_grupa').annotate(
            broj=Count('id'),
            prosecna_vrednost=Avg('vrednost')
        )

# Korišćenje
podaci = Merenje.objects.vremenska_grupa('1 dan')
```

3. ORM kao mehanizam za čuvanje događaja

Implementacija čuvanja događaja kroz ORM:

python

```
class Dogadjaj(Model):
    id_agregata = UUIDField()
    tip_dogadjaja = CharField(max_length=50)
    podaci_dogadjaja = JSONField()
    verzija = IntegerField()
    kreiran = DateTimeField(auto_now_add=True)

    class Meta:
        unique_together = ['id_agregata', 'verzija']

class SkladisteDogadjaja:
    def  dodaj_dogadjaj(self,  id_agregata,  tip_dogadjaja,
podaci, ocekivana_verzija):
        with transaction.atomic():
            dogadjaj = Dogadjaj(
                id_agregata=id_agregata,
                tip_dogadjaja=tip_dogadjaja,
                podaci_dogadjaja=podaci,
                verzija=ocekivana_verzija + 1
            )
            dogadjaj.save()
            return dogadjaj
```

Napredni saveti za kvalitetno korišćenje

1. Strategija lenjog naspram željnog učitavanja

python

```
# Podesite uobičajeno povezano učitavanje za često korišćene
veze
class KnjigaQuerySet(QuerySet):
    def objavljene(self):
        return
self.filter(objavljena=True).select_related('autor')

class Knjiga(Model):
    objects = KnjigaQuerySet.as_manager()
```

2. Prilagođeni tipovi polja za specifične potrebe

python

```python
class SifrovanoPolje(CharField):
    def from_db_value(self, vrednost, izraz, konekcija):
        if vrednost is None:
            return vrednost
        return desifruj(vrednost)

    def to_python(self, vrednost):
        if isinstance(vrednost, str):
            return vrednost
        return desifruj(vrednost)

    def get_prep_value(self, vrednost):
        return sifruj(vrednost)
```

3. Funkcije baze podataka kroz ORM

python

```python
from django.db.models import Func

class Slicnost(Func):
    function = 'SIMILARITY'
    output_field = FloatField()

# Korišćenje za neodređenu pretragu
Korisnik.objects.annotate(
    slicnost=Slicnost('ime', Value('Marko Petrović'))
).filter(slicnost__gt=0.3)
```

Performanse: Praćenje i optimizacija

1. Analiza upita

python

```python
from django.db import connection

def analiziraj_upite():
    print(f"Broj upita: {len(connection.queries)}")
    for upit in connection.queries:
        print(f"Vreme: {upit['time']}, SQL: {upit['sql']}")
```

2. Skupljanje konekcija

python

```
# settings.py
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'OPTIONS': {
            'MAX_CONNS': 20,
            'MIN_CONNS': 5,
        }
    }
}
```

Arhitekturni obrasci sa ORM-om

1. Obrazac skladišta

python

```
class KorisnikSkladiste:
    def __init__(self, model=Korisnik):
        self.model = model

    def nadji_aktivne_korisnike(self):
        return self.model.objects.filter(aktivan=True)

    def nadji_po_domenu_mejla(self, domen):
        return
self.model.objects.filter(mejl__endswith=f'@{domen}')
```

2. Obrazac jedinice rada

python

```
class JedinicaRada:
    def __init__(self):
        self._novi_objekti = []
        self._izmenjeni_objekti = []
        self._uklonjeni_objekti = []

    def registruj_novi(self, obj):
        self._novi_objekti.append(obj)
```

```
    def potvrditi(self):
        with transaction.atomic():
            for obj in self._novi_objekti:
                obj.save()
            for obj in self._izmenjeni_objekti:
                obj.save()
            for obj in self._uklonjeni_objekti:
                obj.delete()
```

Budućnost ORM-a: Pravci razvoja

1. Podrška za asinhrono izvršavanje

Savremene ORM biblioteke sve više podržavaju asinhroni pristup:

python

```
# Django 4.1+
async def dobij_korisnike():
    korisnici = []
    async for korisnik in Korisnik.objects.filter(aktivan=True):
        korisnici.append(korisnik)
    return korisnici
```

2. Bezbednost tipova

Uključivanje sa proverivačima tipova:

python

```
from typing import List
from django_stubs_ext import QuerySetType

class Korisnik(Model):
    ime: str
    mejl: str

    objects: QuerySetType[Korisnik]

# MyPy će proveriti tipove
korisnici: List[Korisnik] = Korisnik.objects.filter(aktivan=True)
```

Zaključak: Ravnoteža između moći i odgovornosti

ORM predstavlja moćan alat koji može značajno poboljšati produktivnost i sigurnost razvoja programa, ali zahteva duboko razumevanje kako svojih mogućnosti tako i ograničenja. Ključ uspešne upotrebe leži u pronalaženju ravnoteže između jednostavnosti korišćenja i kontrole nad performansama.

Napredni programeri trebalo bi da pristupe ORM-u kao alatu koji dopunjuje, a ne zamenjuje znanje SQL-a i razumevanje relacionih baza podataka. Kombinovanjem stvaralačkog pristupa sa čvrstim tehničkim osnovama, ORM može postati temelj za građenje proširljivih i održivih programa.

Kroz neprestano učenje i eksperimentisanje sa naprednim tehnikama, možemo iskoristiti punu moć ORM-a dok izbegavamo česte zamke i loše obrasce koji mogu ugroziti performanse i održivost naših programa.

Tehnologija ne stoji na jednom mestu - ona se razvija u skladu sa našim rastućim razumevanjem složenih odnosa između podataka, objekata i programa koji ih povezuju. ORM je dokaz da mogu postojati elegantna rešenja za naizgled nepomirljive razlike između različitih načina organizovanja informacija.