**Код программы**

Таблица **Place** хранит информацию о местях:

class Place(models.Model):

Name = models.CharField(max\_length= 100, default=' ')

City = models.ForeignKey(City, on\_delete= False, default= 1)

Rating = models.DecimalField(default= 0.0, max\_digits= 3, decimal\_places=1)

Address = models.CharField(max\_length= 100, default= '')

Description = models.TextField(default='')

Type = models.IntegerField(default= 1)

Site = models.CharField(max\_length= 100, default='')

Tel = models.CharField(max\_length=20, default='')

models.UniqueConstraint(fields= ['Name'], name = 'Unique\_Place')

def \_\_str\_\_(self):

return self.Name

def getImg(self):

imgs = self.img\_set.all()

urls = [item.url for item in imgs]

return urls

def getFirstImg(self):

img = Img.objects.all().filter(place = self.id)[0]

return img

def getReview(self):

return Review.objects.all().filter(place = self.id)

def getRating(self):

reviews = self.getReview()

sum = 0

n = len(reviews)

if (n == 0):

return 0

for item in reviews:

sum = sum + item.rating

rating = round(sum / n, 1)

self.Rating = rating

self.save()

return rating

def updateRating(self):

self.Rating = self.getRating()

Таблица **User** хранит информацию о пользователях:

class AbstractUser(AbstractBaseUser, PermissionsMixin):

"""

An abstract base class implementing a fully featured User model with

admin-compliant permissions.

Username and password are required. Other fields are optional.

"""

username\_validator = UnicodeUsernameValidator()

username = models.CharField(

\_('username'),

max\_length=150,

unique=True,

help\_text=\_('Required. 150 characters or fewer. Letters, digits and @/./+/-/\_ only.'),

validators=[username\_validator],

error\_messages={

'unique': \_("A user with that username already exists."),

},

)

first\_name = models.CharField(\_('first name'), max\_length=30, blank=True)

last\_name = models.CharField(\_('last name'), max\_length=150, blank=True)

email = models.EmailField(\_('email address'), blank=True)

is\_staff = models.BooleanField(

\_('staff status'),

default=False,

help\_text=\_('Designates whether the user can log into this admin site.'),

)

is\_active = models.BooleanField(

\_('active'),

default=True,

help\_text=\_(

'Designates whether this user should be treated as active. '

'Unselect this instead of deleting accounts.'

),

)

date\_joined = models.DateTimeField(\_('date joined'), default=timezone.now)

objects = UserManager()

EMAIL\_FIELD = 'email'

USERNAME\_FIELD = 'username'

REQUIRED\_FIELDS = ['email']

class Meta:

verbose\_name = \_('user')

verbose\_name\_plural = \_('users')

abstract = True

def clean(self):

super().clean()

self.email = self.\_\_class\_\_.objects.normalize\_email(self.email)

def get\_full\_name(self):

"""

Return the first\_name plus the last\_name, with a space in between.

"""

full\_name = '%s %s' % (self.first\_name, self.last\_name)

return full\_name.strip()

def get\_short\_name(self):

"""Return the short name for the user."""

return self.first\_name

def email\_user(self, subject, message, from\_email=None, \*\*kwargs):

"""Send an email to this user."""

send\_mail(subject, message, from\_email, [self.email], \*\*kwargs)

class User(AbstractUser):

"""

Users within the Django authentication system are represented by this

model.

Username and password are required. Other fields are optional.

"""

class Meta(AbstractUser.Meta):

swappable = 'AUTH\_USER\_MODEL'

class Profile(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE)

avatar = models.ImageField(null=True, default='./icon-login.png')

def \_\_str\_\_(self):

return self.user.username

@receiver(post\_save, sender=User) # add this

def create\_user\_profile(sender, instance, created, \*\*kwargs):

if created:

Profile.objects.create(user=instance)

@receiver(post\_save, sender=User) # add this

def save\_user\_profile(sender, instance, \*\*kwargs):

instance.profile.save()

def createProfile(sender, \*\*kwargs):

if kwargs['created']:

user\_profile = Profile.objects.created(user=kwargs['instance'])

post\_save.connect(createProfile, sender=User)

Таблица Review хранит информацию о комментариях:

class Review(models.Model):

comment = models.CharField(max\_length=1000)

createTime = models.DateTimeField(timezone.datetime.now())

place = models.ForeignKey(Place, on\_delete=models.CASCADE)

auth = models.ForeignKey(User, on\_delete=models.CASCADE)

rating = models.IntegerField(default=0,

validators= [MaxValueValidator(5),

MinValueValidator(0),]

)

def \_\_str\_\_(self):

return self.comment

Таблица Image хранит информацию о изображениях:

class Img(models.Model):

img = models.ImageField(null=True)

name = models.CharField(max\_length=100, null=True, default= '')

place = models.ForeignKey(Place, on\_delete=models.CASCADE)

def \_\_str\_\_(self):

return self.name

Urls:

urlpatterns = [

path('product=<int:id>/', views.showProduct, name = 'product'),

path('search=<str:name>/', views.search, name = 'search'),

path('contact/', views.showContact, name = 'contact'),

path('profile/', views.editProfile, name = 'profile'),

path('Type=<str:type>&Order=<str:order>&Filter=<str:filter>/', views.showAllProduct, name='type\_order\_filter'),

path('Type=<str:type>&Filter=<str:filter>/', views.showAllProduct, name='type\_filter'),

path('Type=<str:type>&Order=<str:order>/', views.showAllProduct, name='type\_order'),

path('Type=<str:type>/', views.showAllProduct, name='type'),

path('<str:str>/', views.show, name='show'),

]

Triggers

def createCityAfterDeleteTrigger(connection):

name = 'City\_After\_Delete\_Trigger'

c = connection.cursor()

deleteTrigger(connection, name)

command = '''

CREATE TRIGGER {}

BEFORE DELETE ON product\_city

BEGIN

UPDATE product\_place

SET City\_id = -1

WHERE City\_id = OLD.id;

END;

'''.format(name)

c.execute(command)

connection.commit()

print("Created {}".format(name))

def createCheckMailTrigger(connection):

name = 'Check\_Mail\_Trigger'

c = connection.cursor()

deleteTrigger(connection, name)

command = '''

CREATE TRIGGER {}

BEFORE INSERT ON auth\_user

BEGIN

SELECT

CASE

WHEN NEW.name NOT LIKE '%\_@\_\_%.\_\_%' THEN

RAISE (ABORT,'From Trigger Check\_Mail\_Trigger: Invalid email address')

END;

END;

'''.format(name)

c.execute(command)

connection.commit()

def createCheckUsernameTrigger(connection):

name = 'Check\_Username\_Trigger'

c = connection.cursor()

deleteTrigger(connection, name)

command = '''

CREATE TRIGGER {}

BEFORE INSERT ON auth\_user

BEGIN

SELECT

CASE

WHEN EXISTS (SELECT \* FROM product\_city P

WHERE P.name = NEW.name) THEN

RAISE (ABORT,'From Trigger Check\_Username\_Trigger :Invalid Username')

END;

END;

'''.format(name)

c.execute(command)

connection.commit()

def createDeleteProfileTrigger(connection):

name = 'Delete\_Profile\_Trigger'

c = connection.cursor()

deleteTrigger(connection, name)

command = '''

CREATE TRIGGER {}

AFTER DELETE ON product\_profile

BEGIN

DELETE FROM auth\_user

WHERE id = OLD.user;

END;

'''.format(name)

#c.execute(command)

connection.commit()

def createInsertReviewTrigger(connection):

name = 'After\_Insert\_Review\_Trigger'

c = connection.cursor()

deleteTrigger(connection, name)

command = '''

CREATE TRIGGER {}

AFTER INSERT ON product\_place

BEGIN

UPDATE product\_place

SET raiting = (SELECT AVG(rating)

FROM product\_review

WHERE place\_id = NEW.id)

WHERE id = NEW.id;

END;

'''.format(name)

c.execute(command)

connection.commit()