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()
```