Использование ORM

5 модуль

План

- Что такое ORM: для чего, как работает
- Как выглядит Django models
- Пример моделирования отношений
- Как сделать связи многие-ко-многим и один-к-одному
- Запросы

Пример с магазином: товары, категории.

Что такое ORM

Объектно-Реляционное Отображение (*Object-Relational Mapping*) - технология, связывающая модель базы данных и концепции ООП.

Позволяет работать с данными как с объектами.

Преимущества ORM

- Представление модели данных в ORM независимо от СУБД
- Упрощенное моделирование базы данных
- ORM предоставляет больше механизмов обеспечения целостности данных
- Возможность использовать наследование моделей

Недостатки ORM

- Медленее, чем “голый” SQL
- Возможны проблемы с комплексными запросами
- Идея всех ORM схожа, но реализации различны: нужно дополнительное обучение

Django ORM

```
from django.db import models


class Category(models.Model):
    name = models.CharField(max_length=100)

    def get_available_products(self):
        """ Gets products of current category with quantity>0 """
        return Products.objects.filter(category=self, quantity__gte=0)


class Product(models.Model):
    name = models.CharField(max_length=100)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    quantity = models.IntegerField(default=0)
    category = models.ForeignKey(Category, null=True)
```

Миграции

- Упрощают работу со схемой базы
- Позволяют быстро переключаться между разными вариантами схемы (в том числе откатывать изменения)

Миграции базы данных Django

manage.py makemigrations - создает непривязанные к СУБД инструкции для отображения текущего состояния моделей

manage.py migrate - используя миграции приводит базу данных, указанную в *settings.py* к виду актуальному *models.py*

Онлайн-магазин

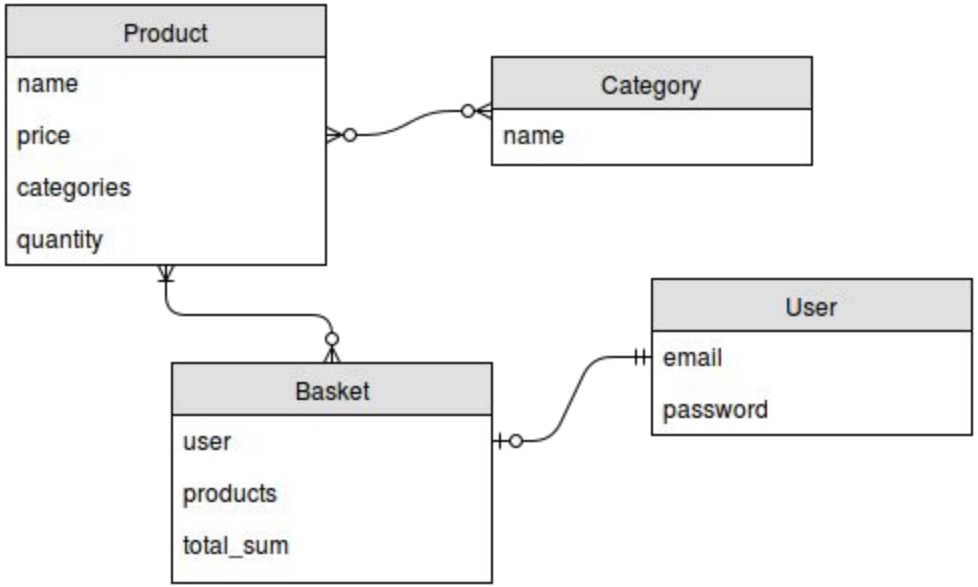
Товар: название, цена, количество, категории

Категория: название

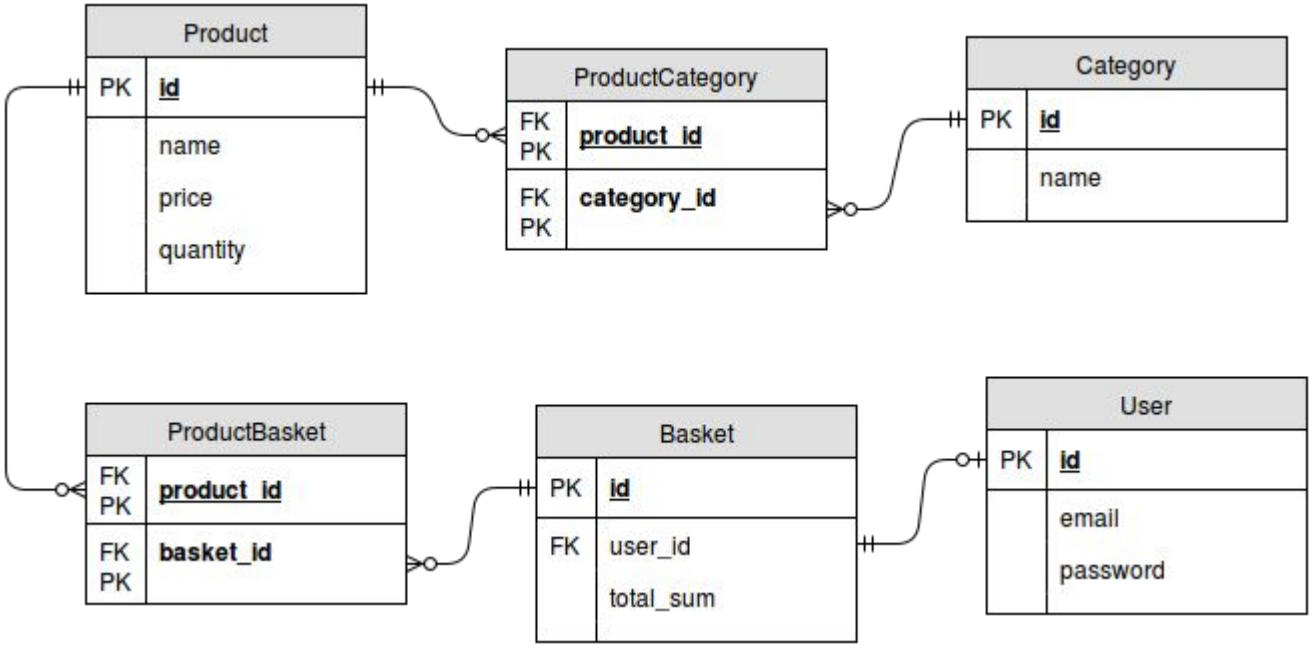
Пользователь: почта, пароль

Корзина: пользователь, товары, итоговая сумма

ER-диаграмма



Нормализована схема



Модели в Django

```
from django.db import models
from django.contrib.auth.models import User


class Category(models.Model):
    name = models.CharField(max_length=100)


class Product(models.Model):
    name = models.CharField(max_length=100)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    quantity = models.IntegerField(default=0)
    categories = models.ManyToManyField(Category)


class Basket(models.Model):
    user = models.OneToOneField(User)
    products = models.ManyToManyField(Category, null=True)
    total_sum = models.DecimalField(max_digits=10, decimal_places=2)
```

Созданные таблицы

- auth_user
- catalog_category
- catalog_basket
- catalog_basket_products
- catalog_product
- catalog_product_categories

ManyToManyField

```
sqlite> .schema catalog_product_categories

CREATE TABLE "catalog_product_categories" (
        "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
        "product_id" integer NOT NULL REFERENCES "catalog_product" ("id"),
        "category_id" integer NOT NULL REFERENCES "catalog_category" ("id")
);

CREATE UNIQUE INDEX "catalog_product_categories_product_id_9802d7aa_uniq" ON
"catalog_product_categories" ("product_id", "category_id");
CREATE INDEX "catalog_product_categories_9bea82de" ON "catalog_product_categories"
("product_id");
CREATE INDEX "catalog_product_categories_b583a629" ON "catalog_product_categories"
("category_id");
```

OneToOneField

```
sqlite> .schema catalog_basket
CREATE TABLE "catalog_basket" (
        "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
        "total_sum" decimal NOT NULL,
        "user_id" integer NOT NULL UNIQUE REFERENCES "auth_user" ("id")
);
```