AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA INFORMATYKI

BAZY DANYCH

Technologia: **Python + Django**

Serwer BD: **MS Sql**

Repository: *https://github.com/robertcialowicz/bazydanych2.git*

Autorzy:

Chwała Paweł <pchwala@student.agh.edu.pl>

Ciałowicz Robert <robcial@student.agh.edu.pl>

Kozaczkiewicz Łukasz <kozaczki@student.agh.edu.pl>

Szpila Magdalena <mszpila@student.agh.edu.pl>

Kraków, 2020

Spis treści

Opis projektu

Aplikacja napisana w języku Python z wykorzystaniem Django Rest Framework.

W projekcie wykorzystano serwer bazy danych MSSql.

Przykładową bazą, na której wykonywane będą operacje jest baza Northwind.

Celem projektu jest zaimplementowanie operacji CRUD na dowolnej tabeli, operacji składania zamówienia oraz możliwość tworzenia raportów (do zdefiniowania).

Struktura projektu i uruchomienie

W lokalizacji `./src` znajduje się plik `docker-compose.yml`. Aby wystartować aplikację należy z konsoli wywołać `docker-compose up –build`.

Uruchomią się dwa kontenery dockerowe zdefiniowane w pliku `yml`: jeden z serwerem MSSql z bazą danych Northwind oraz drugi z aplikacją pythonową.

Na porcie lokalnym 8000 zostanie uruchomiona aplikacja Django.

Dane do logowania admin/admin.

UWAGA!

W obecnej wersji aplikacja nie działa na systemach platformy Windows.

Struktura projektu:

src/sqlserver - konfiguracja dockera udostępniającego serwer sql

src/web - konfiguracja dockera udostępniającego aplikację webową

src/myapp, myapi - aplikacje oparte na frameworku Django wchodzące w skład projektu

src/manage.py - Narzędzie linii komend, które pozwala oddziaływać z projektem Django. Więcej szczegółów na temat manage.py zostało opisanych w punkcie Projekt Django

src/inspect.db - plik zawierający automatycznie wygenerowane modele na podstawie bazy danych Northwind. Został wygenerowany z wykorzystaniem manage.py i komendy `inspectdb.` Pełni funkcje jedynie poglądowe i został zachowany aby zaprezentować narzędzia wspomagające integrowanie aplikacji w Django z “odziedziczonymi” bazami danych.

src/docker-compose.yml - konfiguracja dla uruchomienia wszystkich składowych projektu

Serwer Microsoft SQL

Uruchomienie

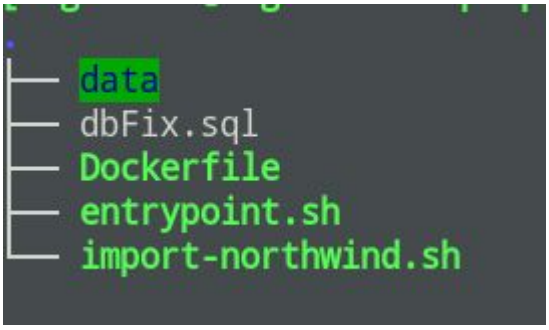
Skorzystaliśmy z gotowego obrazu dockerowego Microsoft SQL Server 2019. Najprostszym sposobem na uruchomienie serwera bazy danych jest wykonanie komendy z poziomu katalogu /src

```
docker-compose -up --build sqlserver
```

Po starcie serwera najpierw jest importowana baza Northwind, a następnie nakładana jest na to nasza poprawka.

Struktura projektu

Zawartość katalogu **src/sqlserver**



import-northwind.sh - skrypt uruchamiający serię komend SQL-owych pozwalający utworzyć i spopulować bazę Northwind - jeżeli nie istnieje na serwerze MS SQL

entrypoint.sh - skrypt uruchamiający serwer MS SQL a następnie wywołujący skrypt import-northwind.sh wewnątrz wystartowanego dockera

dbFix.sh - skrypt wywołujący serię koment SQL-owych modyfikujących tabele w bazie zgodnie z wymaganiami frameworka Django

data - wolumen z danymi bazy, obecnie “nie podpinany”

Dockerfile - plik z konfiguracją dockera

Skrypt naprawczy

Na potrzeby projektu musieliśmy dokonać pewnych zmian w bazie danych. Było to spowodowane tym, że Django wymaga, aby każda tabela miała dokładnie jeden klucz główny. W bazie Northwind tabela *OrderDetails* nie ma takiego klucza. Poniżej zamieszczamy skrypt, który dodaje brakujący klucz.

```
USE [Northwind]
GO
ALTER TABLE [Order Details] DROP CONSTRAINT IF EXISTS pk_order_details
ALTER TABLE [Order Details] DROP CONSTRAINT IF EXISTS uk_order_details
ALTER TABLE [Order Details] DROP COLUMN IF EXISTS orderdetailid
ALTER TABLE [Order Details] ADD orderdetailid INTEGER IDENTITY(1,1) NOT NULL
GO
ALTER TABLE [Order Details] ADD CONSTRAINT pk_order_details PRIMARY KEY (orderdetailid)
ALTER TABLE [Order Details] ADD CONSTRAINT uk_order_details UNIQUE (orderid, productid)
```

Projekt django

Jak zacząć

Webowy projekt dla poprawnego działania wymaga możliwości połączenia do bazy danych co zapewnia konfiguracja w docker-compose.yml. Najlepszym sposobem na uruchomienie aplikacji jest wykonanie komendy z poziomu katalogu /src

```
docker compose -up --build web
```

Przy starcie dockera instalowane są potrzebne biblioteki do pracy z frameworkiem Django i bazą danych, kopiowany jest kod projektu do wnętrza kontenera a następnie uruchamiany jest skrypt wait-for-it.sh, którego zadaniem jest wstrzymanie uruchomienia aplikacji Django do momentu aż dostępny będzie host i port serwera MS SQL. Projekt Django uruchamiany jest komendą

```
python3 manage.py runserver 0.0.0.0:8000
```

manage.py - Narzędzie linii komend, które pozwala oddziaływać z tym projektem Django na wiele sposobów. Automatycznie generowany w każdym projekcie Django. Z najważniejszych udostępnianych komend warto zwrócić uwagę na:

`makemigrations` - tworzy nowe migracje na podstawie zmian wykrytych w modelu

`migrate` - synchronizuje aktualny stan bazy danych ze stanem modelu i migracji

`createsuperuser` - usługa dostępna przy zainstalowanym pakiecie `django.contrib.auth`, tworzy konto użytkownika z pełnymi prawami dostępu. Przydatne przy rozpoczynaniu pracy z modułem `admin`

`runserver` - uruchamia lekki serwer webowy na lokalnej maszynie

Generowanie modelu

Aby wygenerować model bazy Northwind użyto funkcji inspectdb frameworku Django. Poszczególne kroki przedstawiają się następująco:

1. Uruchomienie aplikację - python src/manage.py startapp myapp
2. Wygenerowanie i zapisanie modelu - python src/manage.py inspectdb > src/myapi/models.py
3. Naniesienie poprawek na model:
    - w przypadku relacji many-to-many (Employees Territories, Order Details, CustomerCustomerDemo) usunięto ManyToManyField z tabel tworzących te relacje
    - zmieniono kolejność klas w modelu zgodnie z odwołaniami. Końcowa kolejność to: Categories, Shippers, Customercustomerdemo, CustomerDeographics, Customer, Region, Territories, Employees, Suppliers, Products, Orders, OrderDetails, Employeeterritories
    - w tabelach, które nie miały klucza głównego ustawiono odpowiedni klucz główny, poprzez primary_key = True

Pomimo faktu, że framework django jest polecany przede wszystkim do green-field development czyli budowania projektu od zera, w przypadku bazy Northwind wygenerowanie modelu i poprawki było relatywnie proste. Możliwość autogeneracji modelu przez framework jest ogromną zaletą gdyż pozwala oszczędzić czas oraz uniknąć pomyłek. Należy jednak pamiętać, że takie podejście może doprowadzić do niepożądanych zachowań. W trakcie pracy nad tym projektem takich sytuacji nie zaobserwowano

Django admin

Panel Django admin jest to interface użytkownika dostarczany przez framework, umożliwiający przygotowanie w szybki sposób uproszczonego front endu na potrzeby developmentu i testowania aplikacji. Po uruchomieniu serwera django poprzez python manage.py runserver panel ten jest dostępny domyślnie pod localhost:8000/admin

W przypadku istniejących obiektów, np klasa product:

w pliku myapi/admin.py należy

1. zaimportować tę klasę: from .models import Products
2. zarejestrować ją w panelu:admin.site.register(Products)

Dostarczanie przez framework panelu admin jest bardzo pomocna, gdyż przy minimalnym nakładzie pracy umożliwia w prosty sposób testowanie funkcjonalności aplikacji.

Domyślne zachowanie zaimplementowane w ten sposób, nie zwraca jednak zbyt wiele informacji o obiekcie. Zamiast nazw wyświetlane są jedynie id obiektów referujących przez klucze obce. Aby zmienić to zachowanie należy w pliku models.py nadpisać funkcję __str__(self). W ten sposób możemy reprezentować obiekt przez dowolnie zdefiniowany ciąg znaków. W przypadku tego projektu dla klasy Products jest to:

```
def __str__(self):

        return self.productname
```

Dla pozostałych klas w modelu zaimplementowano analogiczne rozwiązanie.

Warto zauważyć, że to rozwiązanie nie zmienia nic w wydajności programu. Do serwera SQL wysyłane są dokładnie te same zapytania. Oznacza to, że django automatycznie pobiera z bazy całe obiekty, nawet jeśli tylko referują one przez klucz obcy.

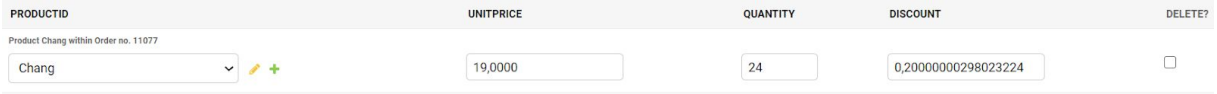
```
QUERY = 'SELECT [Products].[ProductID], [Products].[ProductName], [Products].[SupplierID], [Products].[CategoryID],
[Products].[QuantityPerUnit], [Products].[UnitPrice], [Products].[UnitsInStock], [Products].[UnitsOnOrder],
[Products].[ReorderLevel], [Products].[Discontinued] FROM [Products] WHERE [Products].[ProductID] = %s' - PARAMS =
(76,); args=(76,)

QUERY = 'SELECT [Suppliers].[SupplierID], [Suppliers].[CompanyName], [Suppliers].[ContactName],
[Suppliers].[ContactTitle], [Suppliers].[Address], [Suppliers].[City], [Suppliers].[Region],
[Suppliers].[PostalCode], [Suppliers].[Country], [Suppliers].[Phone], [Suppliers].[Fax], [Suppliers].[HomePage] FROM
[Suppliers]' - PARAMS = (); args=()

QUERY = 'SELECT [Categories].[CategoryID], [Categories].[CategoryName], [Categories].[Description],
[Categories].[Picture] FROM [Categories]' - PARAMS = (); args=()
```

Ponadto panel Django admin umożliwia dodanie formularza z obiektami wbudowanymi (inline), wtedy w jednym widoku widoczne są obiekty danego typu, ale także obiekty referujące do nich przez klucz obcy. Poniżej przykład z projektu, gdzie obiektem nadrzędnym jest obiekt Product, a obiekty wbudowane to obiekty typu OrderDetails.

```
class ProductsAdmin(admin.ModelAdmin):

    inlines = (OrderDetailsInline, )

admin.site.register(Products, ProductsAdmin)
```

Warto zwrócić uwagę na to, że zmienił się sposób rejestrowania obiektu.

Wykorzystując powyższą instrukcję jesteśmy w stanie wykonywać podstawowe operacje na obiektach zależnych, czyli w naszym przypadku produktach. Możemy je dodawać, usuwać i edytować w bardzo prosty sposób z poziomu podglądu zamówienia. Dostajemy tą funkcjonalność “za darmo” w postaci odpowiednich przycisków obok każdego z produktów.



Wprowadzenie obiektów typu inline wprowadza również zmiany w zapytaniach SQL wysyłanych do serwera. Dla obiektu Products bez wbudowanych obiektów zapytanie wyglądają jak te przedstawione powyżej. Dla obiektu Products z wbudowanymi obiektami typu OrderDetails zapytań znacznie przybywa.

```
QUERY = 'SELECT [Products].[ProductID], [Products].[ProductName], [Products].[SupplierID], [Products].[CategoryID],
[Products].[QuantityPerUnit], [Products].[UnitPrice], [Products].[UnitsInStock], [Products].[UnitsOnOrder],
[Products].[ReorderLevel], [Products].[Discontinued] FROM [Products] WHERE [Products].[ProductID] = %s' - PARAMS =
(76,); args=(76,)

QUERY = 'SELECT [Order Details].[OrderID], [Order Details].[ProductID], [Order Details].[UnitPrice], [Order
Details].[Quantity], [Order Details].[Discount], [Order Details].[orderdetailid] FROM [Order Details] WHERE [Order
Details].[ProductID] = %s ORDER BY [Order Details].[orderdetailid] ASC' - PARAMS = (76,); args=(76,)

QUERY = 'SELECT [Suppliers].[SupplierID], [Suppliers].[CompanyName], [Suppliers].[ContactName],
[Suppliers].[ContactTitle], [Suppliers].[Address], [Suppliers].[City], [Suppliers].[Region],
[Suppliers].[PostalCode], [Suppliers].[Country], [Suppliers].[Phone], [Suppliers].[Fax], [Suppliers].[HomePage] FROM
[Suppliers]' - PARAMS = (); args=()

QUERY = 'SELECT [Categories].[CategoryID], [Categories].[CategoryName], [Categories].[Description],
[Categories].[Picture] FROM [Categories]' - PARAMS = (); args=()

QUERY = 'SELECT [Products].[ProductID], [Products].[ProductName], [Products].[SupplierID], [Products].[CategoryID],
[Products].[QuantityPerUnit], [Products].[UnitPrice], [Products].[UnitsInStock], [Products].[UnitsOnOrder],
[Products].[ReorderLevel], [Products].[Discontinued] FROM [Products] WHERE [Products].[ProductID] = %s' - PARAMS =
(76,); args=(76,)

QUERY = 'SELECT [Orders].[OrderID], [Orders].[CustomerID], [Orders].[EmployeeID], [Orders].[OrderDate],
[Orders].[RequiredDate], [Orders].[ShippedDate], [Orders].[ShipVia], [Orders].[Freight], [Orders].[ShipName],
[Orders].[ShipAddress], [Orders].[ShipCity], [Orders].[ShipRegion], [Orders].[ShipPostalCode], [Orders].[ShipCountry]
FROM [Orders] WHERE [Orders].[OrderID] = %s' - PARAMS = (10343,); args=(10343,)

QUERY = 'SELECT [Orders].[OrderID], [Orders].[CustomerID], [Orders].[EmployeeID], [Orders].[OrderDate],
[Orders].[RequiredDate], [Orders].[ShippedDate], [Orders].[ShipVia], [Orders].[Freight], [Orders].[ShipName],
[Orders].[ShipAddress], [Orders].[ShipCity], [Orders].[ShipRegion], [Orders].[ShipPostalCode], [Orders].[ShipCountry]
FROM [Orders]' - PARAMS = (); args=()

QUERY = 'SELECT [Products].[ProductID], [Products].[ProductName], [Products].[SupplierID], [Products].[CategoryID],
[Products].[QuantityPerUnit], [Products].[UnitPrice], [Products].[UnitsInStock], [Products].[UnitsOnOrder],
[Products].[ReorderLevel], [Products].[Discontinued] FROM [Products] WHERE [Products].[ProductID] = %s' - PARAMS =
(76,); args=(76,)

QUERY = 'SELECT [Orders].[OrderID], [Orders].[CustomerID], [Orders].[EmployeeID], [Orders].[OrderDate],
[Orders].[RequiredDate], [Orders].[ShippedDate], [Orders].[ShipVia], [Orders].[Freight], [Orders].[ShipName],
[Orders].[ShipAddress], [Orders].[ShipCity], [Orders].[ShipRegion], [Orders].[ShipPostalCode], [Orders].[ShipCountry]
FROM [Orders] WHERE [Orders].[OrderID] = %s' - PARAMS = (10267,); args=(10267,)

QUERY = 'SELECT [Orders].[OrderID], [Orders].[CustomerID], [Orders].[EmployeeID], [Orders].[OrderDate],
[Orders].[RequiredDate], [Orders].[ShippedDate], [Orders].[ShipVia], [Orders].[Freight], [Orders].[ShipName],
[Orders].[ShipAddress], [Orders].[ShipCity], [Orders].[ShipRegion], [Orders].[ShipPostalCode], [Orders].[ShipCountry]
FROM [Orders]' - PARAMS = (); args=()

...
```

Widać, że dla każdego obiektu zależnego wysyłane są 3 dodatkowe zapytania SQL. W przypadku dużej ilości obiektów powiązanych, znacząco wpływa to na czas ładowania widoku pojedynczego produktu. Przykładowo dla Produktu Lakkalikööri czas wzrasta z około 800 ms do 17 sekund.

Dla czytelności panelu administracyjnego usunięto domyślne formularze dla użytkowników i grup za pomocą metody *unregister:*

```
admin.site.unregister(User)
admin.site.unregister(Group)
```

Dostosowanie formularzy

Walidacja pól formularza znajduje się w metodach *clean* klas *OrdersForm* oraz *OrderDetailsInlineFormSet* w zależności od tego czy pracujemy z obiektami typu wbudowanego czy nie. W metodzie clean możemy natomiast dowolnie zdefiniować swoje kryteria oraz treść zwracanego wyjątku.

Dla zamówienia sprawdzamy czy data zamówienia (*OrderDate*) jest co najmniej dzień przed datą zapotrzebowania (*RequiredDate*).

```
def clean(self):
        orderDate = self.cleaned_data.get('orderdate')
        requiredDate = self.cleaned_data.get('requireddate')
        if orderDate >= (requiredDate - datetime.timedelta(days=1)):
            raise forms.ValidationError("Orderdate must be at least 24 hours before
Required date!")
        return self.cleaned_data
```

Natomiast dla produktu sprawdzamy czy nie został dodany dwa razy do tego samego zamówienia, czy jego cena za jednostkę jest większa od 0, czy zniżka jest w zakresie [0, 1] oraz czy w magazynie znajduje się wystarczająca ilość tego produktu. Kod walidacji produktu znajduje się poniżej.

```
def clean(self):
        setOfProducts = set()
        if(self.is_valid()):
            for productForm in self.cleaned_data:
                reservedQuantity = productForm.get('quantity')
                product = productForm.get('productid')
                unitsInStock = product.unitsinstock
                price = productForm.get('unitprice')
                discount = productForm.get('discount')
                if product in setOfProducts:
                    raise forms.ValidationError("Product " + str(product) + " was added
more than once!")
                else:
                    setOfProducts.add(product)
                if price <= 0:
                    raise forms.ValidationError("Unitprice for product " + str(product) +
" has to be greater than 0!")
                if discount < 0 or discount > 1:
                    raise forms.ValidationError("Discount for product " + str(product) +
" has to be value between 0 and 1!")
                if reservedQuantity > unitsInStock:
                    raise forms.ValidationError("Maximum quantity for product " +
str(product) + " is " + str(unitsInStock) + "!")
```

Dodanie do zamówienia listy przypisanych do niego produktów powoduje wykonanie dodatkowych zapytań do bazy danych, co z kolei ma wpływ na szybkość ładowania się strony. Przed dodaniem do formularza listy produktów po wejściu w edycję istniejącego zamówienia były wykonywane poniższe zapytania. Zapytania przed dodaniem listy produktów do widoku zamówienia:

```
SELECT [Orders].[OrderID], [Orders].[CustomerID], [Orders].[EmployeeID], [Orders].[OrderDate],
[Orders].[RequiredDate], [Orders].[ShippedDate], [Orders].[ShipVia], [Orders].[Freight], [Orders].[ShipName],
[Orders].[ShipAddress], [Orders].[ShipCity], [Orders].[ShipRegion], [Orders].[ShipPostalCode],
[Orders].[ShipCountry] FROM [Orders] WHERE [Orders].[OrderID] = 11079;

SELECT [Customers].[CustomerID], [Customers].[CompanyName], [Customers].[ContactName], [Customers].[ContactTitle],
[Customers].[Address], [Customers].[City], [Customers].[Region], [Customers].[PostalCode], [Customers].[Country],
[Customers].[Phone], [Customers].[Fax] FROM [Customers];

SELECT [Employees].[EmployeeID], [Employees].[LastName], [Employees].[FirstName], [Employees].[Title],
[Employees].[TitleOfCourtesy], [Employees].[BirthDate], [Employees].[HireDate], [Employees].[Address],
[Employees].[City], [Employees].[Region], [Employees].[PostalCode], [Employees].[Country], [Employees].[HomePhone],
[Employees].[Extension], [Employees].[Photo], [Employees].[Notes], [Employees].[ReportsTo], [Employees].[PhotoPath]
FROM [Employees];

SELECT [Shippers].[ShipperID], [Shippers].[CompanyName], [Shippers].[Phone] FROM [Shippers];
```

Natomiast po dodaniu listy produktów ilość wykonywanych zapytań wzrosła dwukrotnie. Na szczęście czas ładowania strony wzrósł tylko nieznacznie. Całość zajęła średnio około 600ms, a na odpowiedź serwera czekaliśmy 200ms. Zapytania po dodaniu listy produktów do widoku zamówienia:

```
SELECT [Orders].[OrderID], [Orders].[CustomerID], [Orders].[EmployeeID], [Orders].[OrderDate],
[Orders].[RequiredDate], [Orders].[ShippedDate], [Orders].[ShipVia], [Orders].[Freight], [Orders].[ShipName],
[Orders].[ShipAddress], [Orders].[ShipCity], [Orders].[ShipRegion], [Orders].[ShipPostalCode],
[Orders].[ShipCountry] FROM [Orders] WHERE [Orders].[OrderID] = 11079;

SELECT [Order Details].[OrderID], [Order Details].[ProductID], [Order Details].[UnitPrice], [Order
Details].[Quantity], [Order Details].[Discount], [Order Details].[orderdetailid] FROM [Order Details] WHERE [Order
Details].[OrderID] = 11079 ORDER BY [Order Details].[orderdetailid] ASC;

SELECT [Customers].[CustomerID], [Customers].[CompanyName], [Customers].[ContactName], [Customers].[ContactTitle],
[Customers].[Address], [Customers].[City], [Customers].[Region], [Customers].[PostalCode], [Customers].[Country],
[Customers].[Phone], [Customers].[Fax] FROM [Customers];

SELECT [Employees].[EmployeeID], [Employees].[LastName], [Employees].[FirstName], [Employees].[Title],
[Employees].[TitleOfCourtesy], [Employees].[BirthDate], [Employees].[HireDate], [Employees].[Address],
[Employees].[City], [Employees].[Region], [Employees].[PostalCode], [Employees].[Country], [Employees].[HomePhone],
[Employees].[Extension], [Employees].[Photo], [Employees].[Notes], [Employees].[ReportsTo], [Employees].[PhotoPath]
FROM [Employees];

SELECT [Shippers].[ShipperID], [Shippers].[CompanyName], [Shippers].[Phone] FROM [Shippers];

SELECT [Order Details].[OrderID], [Order Details].[ProductID], [Order Details].[UnitPrice], [Order
Details].[Quantity], [Order Details].[Discount], [Order Details].[orderdetailid] FROM [Order Details] WHERE [Order
Details].[OrderID] = 11079;

SELECT [Products].[ProductID], [Products].[ProductName], [Products].[SupplierID], [Products].[CategoryID],
[Products].[QuantityPerUnit], [Products].[UnitPrice], [Products].[UnitsInStock], [Products].[UnitsOnOrder],
[Products].[ReorderLevel], [Products].[Discontinued] FROM [Products] WHERE [Products].[ProductID] = 21;

SELECT [Orders].[OrderID], [Orders].[CustomerID], [Orders].[EmployeeID], [Orders].[OrderDate],
[Orders].[RequiredDate], [Orders].[ShippedDate], [Orders].[ShipVia], [Orders].[Freight], [Orders].[ShipName],
[Orders].[ShipAddress], [Orders].[ShipCity], [Orders].[ShipRegion], [Orders].[ShipPostalCode],
[Orders].[ShipCountry] FROM [Orders] WHERE [Orders].[OrderID] = 110791;

SELECT [Products].[ProductID], [Products].[ProductName], [Products].[SupplierID], [Products].[CategoryID],
[Products].[QuantityPerUnit], [Products].[UnitPrice], [Products].[UnitsInStock], [Products].[UnitsOnOrder],
[Products].[ReorderLevel], [Products].[Discontinued] FROM [Products];

SELECT [Products].[ProductID], [Products].[ProductName], [Products].[SupplierID], [Products].[CategoryID],
```

```
[Products].[QuantityPerUnit], [Products].[UnitPrice], [Products].[UnitsInStock], [Products].[UnitsOnOrder],
[Products].[ReorderLevel], [Products].[Discontinued] FROM [Products];
```

Pod każdym zamówieniem umieściliśmy dodatkowo krótkie podsumowanie. Zostało ono dodane poprzez dodanie do klasy modelu *Order* property *summary*, a następnie rozdzielenia formularza na dwa formsety. Pierwszy z nich zawiera wszystkie pola z modelu *Order*, a drugi tylko *summary*. Dzięki temu są one wyświetlane osobno. *Summary* wylicza całkowitą wartość zamówienia sumując dla każdego produktu w zamówieniu jego cenę używając poniższego wzoru:

$$X = P * Q * (1 - D)$$

gdzie:

X - ostateczna cena produktu

P - cena jednej sztuki produktu

Q - ilość produktów w zamówieniu

D - zniżka z przedziału [0, 1]

Zdefiniowanie dodatkowej property w pliku models.py:

```
class Orders(models.Model)

    def summary(self):

        sum = 0

        for item in self.orderdetailsFK.through.objects.filter(orderid=self.orderid):

            sum += float(item.unitprice) * float(item.quantity) * (1 -
float(item.discount))

        return "${0}".format(round(sum, 2))
```

Kod odpowiedzialny za zdefiniowanie formsetów z pliku *admin.py* znajduje się poniżej.

```
class OrdersAdmin(admin.ModelAdmin):

    form = OrdersForm

    inlines = (OrderDetailsInline, )

    fieldsets = [

        (None, {'fields': ['customerid', 'employeeid', 'orderdate', 'requireddate',
'shippeddate', 'shipvia', 'freight', 'shipname', 'shipaddress', 'shipcity', 'shipregion',
'shippostalcode', 'shipcountry', ]}),

        ('Order Summary', {'fields': ['summary', ]}),

    ]

    readonly_fields = ('summary', )
```

Tworzenie własnych widoków / modeli / raportów

Poza tworzeniem widoków bazujących na klasach bazowych z pliku models.py, narzędzie Django admin umożliwia tworzenie własnych klas, z dowolnie zdefiniowanymi polami.

W projekcie stworzono klasą o nazwie *Orders Reports*. Jest to klasa, w której widok bazuje na modelu *Orders*, jednak dodano do niego dodatkowe pola.

```
class OrdersReportsAdmin(admin.ModelAdmin):
    list_display = ("orderid", 'customerid', 'orderdate', 'getproducts', 'getcategories', '
getsuppliers')
    list_filter = ("orderdetailsFK__categoryid", "orderdetailsFK__supplierid")
```

Pola *getproducts*, *getcategories* oraz *getsuppliers* to pola, które nie występują bezpośrednio w obiekcie zamówienia. Zdefiniowano je w klasie nadrzędnej, poprzez odniesienie do obiektów referowanych przez klucz obcy.

```
class OrdersReports(Orders):
    class Meta:
        verbose_name_plural = 'Orders Reports'
        proxy = True

    def getproducts(self):
        return ", ".join([
            product.productname for product in self.orderdetailsFK.all()
        ])
    getproducts.short_description = "Products"

    def getcategories(self):
        return ", ".join([
            product.categoryid.categoryname for product in self.orderdetailsFK.all()
        ])
    getcategories.short_description = "Categories"

    def getsuppliers(self):
        return ", ".join([
            product.supplierid.companyname for product in self.orderdetailsFK.all()
        ])
    getsuppliers.short_description = "Suppliers"
```

W dwóch powyższych fragmentach kodu warto zwrócić uwagę na kilka nowych rzeczy. Pierwszą z nich jest lista list_*filter*. To lista z polami, według których będą filtrowane zamówienia. Warto zwrócić uwagę, że wybrane w przykładzie pola filtrowania zamówień nie są polami klasy bazowej Order. Odniesiono się do nich poprzez klucz obcy *OrderDetails*. W tym przypadku *orderdetailsFK* reprezentuje obiekt typu *OrderDetails*, a *orderdetailsFK__categoryid* odnosi się już do konkretnego pola klasy *OrderDetails* (np. *categoryid*).

Kolejną rzeczą na jaką należy zwrócić uwagę jest definicja pól *getproducts*, *getcategories* oraz *getsuppliers*. Są to pola referujące do *Orders* przez klucze obce różnych tabel. Dla przykładu, bazując na właściwości *getcategories*, znalezienie relacji pomiędzy *Orders* i *Categories* wymagała przejścia przez tabele Products oraz *OrderDetails*.

Uruchomienie widoku Orders Reports, zdefiniowanego w powyższy sposób, daje następujące efekt.

ORDERID	CUSTOMERID	ORDERDATE	PRODUCTS	CATEGORIES	SUPPLIERS
11077	Rattlesnake Canyon Grocery	May 6, 1998, midnight	Aniseed Syrup, Chef Anton's Cajun Seasoning, Grandma's Boysenberry Spread, Uncle Bob's Organic Dried Pears, Northwoods Cranberry Sauce, Ikura, Queso Manchego La Pastora, Konbu, Tofu, Pavlova, Sir Rodney's Marmalade, Tunnbröd, Mascarpone Fabioli, Chartreuse verte, Jack's New England Clam Chowder, Spegesild, Filo Mix, Pâté chinois, Camembert Pierrot, Wimmers gute Semmelknödel, Louisiana Hot Spiced Okra, Röd Kaviar, Rhönbräu Klosterbier, Original Frankfurter grüne Soße	Condiments, Condiments, Condiments, Produce, Condiments, Seafood, Dairy Products, Seafood, Produce, Confections, Confections, Grains/Cereals, Dairy Products, Beverages, Seafood, Seafood, Grains/Cereals, Meat/Poultry, Dairy Products, Grains/Cereals, Condiments, Seafood, Beverages, Condiments	Exotic Liquids, New Orleans Cajun Delights, Grandma Kelly's Homestead, Grandma Kelly's Homestead, Grandma Kelly's Homestead, Tokyo Traders, Cooperativa de Quesos 'Las Cabras', Mayumi's, Mayumi's, Pavlova, Ltd., Specialty Biscuits, Ltd., PB Knäckebröd AB, Formaggi Fortini s.r.l., Aux joyeux ecclésiastiques, New England Seafood Cannery, Lyngbysild, G'day, Mate, Ma Maison, Gai pâturage, Plutzer Lebensmittelgroßmärkte AG, New Orleans Cajun Delights, Svensk Sjöföda AB, Plutzer Lebensmittelgroßmärkte AG, Plutzer Lebensmittelgroßmärkte AG
11076	Bon app'	May 6, 1998, midnight	Grandma's Boysenberry Spread, Tofu, Teatime Chocolate Biscuits	Condiments, Produce, Confections	Grandma Kelly's Homestead, Mayumi's, Specialty Biscuits, Ltd.
11075	Richter Supermarkt	May 6, 1998, midnight	Chang, Spegesild, Lakkalikööri	Beverages, Seafood, Beverages	Exotic Liquids, Lyngbysild, Karkki Oy
11074	Simons bistro	May 6, 1998, midnight	Pavlova	Confections	Pavlova, Ltd.

Warto również zwrócić uwagę na zapytania generowane przez framework do serwera SQL. I tak dla jednego wiersza, dla zamówienia nr 11076 otrzymujemy:

```
QUERY = 'SELECT [Customers].[CustomerID], [Customers].[CompanyName], [Customers].[ContactName], [Custom
ers].[ContactTitle], [Customers].[Address], [Customers].[City], [Customers].[Region], [Customers].[Post
alCode], [Customers].[Country], [Customers].[Phone], [Customers].[Fax] FROM [Customers] WHERE [Customer
s].[CustomerID] = %s' - PARAMS = ('BONAP',); args=('BONAP',)
QUERY = 'SELECT [Products].[ProductID], [Products].[ProductName], [Products].[SupplierID], [Products].[
CategoryID], [Products].[QuantityPerUnit], [Products].[UnitPrice], [Products].[UnitsInStock], [Products
].[UnitsOnOrder], [Products].[ReorderLevel], [Products].[Discontinued] FROM [Products] INNER JOIN [Orde
r Details] ON ([Products].[ProductID] = [Order Details].[ProductID]) WHERE [Order Details].[OrderID] =
%s' - PARAMS = (11076,); args=(11076,)
QUERY = 'SELECT [Products].[ProductID], [Products].[ProductName], [Products].[SupplierID], [Products].[
CategoryID], [Products].[QuantityPerUnit], [Products].[UnitPrice], [Products].[UnitsInStock], [Products
].[UnitsOnOrder], [Products].[ReorderLevel], [Products].[Discontinued] FROM [Products] INNER JOIN [Orde
r Details] ON ([Products].[ProductID] = [Order Details].[ProductID]) WHERE [Order Details].[OrderID] =
%s' - PARAMS = (11076,); args=(11076,)
QUERY = 'SELECT [Categories].[CategoryID], [Categories].[CategoryName], [Categories].[Description], [Ca
tegories].[Picture] FROM [Categories] WHERE [Categories].[CategoryID] = %s' - PARAMS = (2,); args=(2,)
QUERY = 'SELECT [Categories].[CategoryID], [Categories].[CategoryName], [Categories].[Description], [Ca
tegories].[Picture] FROM [Categories] WHERE [Categories].[CategoryID] = %s' - PARAMS = (7,); args=(7,)
QUERY = 'SELECT [Categories].[CategoryID], [Categories].[CategoryName], [Categories].[Description], [Ca
tegories].[Picture] FROM [Categories] WHERE [Categories].[CategoryID] = %s' - PARAMS = (3,); args=(3,)
QUERY = 'SELECT [Products].[ProductID], [Products].[ProductName], [Products].[SupplierID], [Products].[
CategoryID], [Products].[QuantityPerUnit], [Products].[UnitPrice], [Products].[UnitsInStock], [Products
].[UnitsOnOrder], [Products].[ReorderLevel], [Products].[Discontinued] FROM [Products] INNER JOIN [Orde
r Details] ON ([Products].[ProductID] = [Order Details].[ProductID]) WHERE [Order Details].[OrderID] =
%s' - PARAMS = (11076,); args=(11076,)
QUERY = 'SELECT [Suppliers].[SupplierID], [Suppliers].[CompanyName], [Suppliers].[ContactName], [Suppli
ers].[ContactTitle], [Suppliers].[Address], [Suppliers].[City], [Suppliers].[Region], [Suppliers].[Post
alCode], [Suppliers].[Country], [Suppliers].[Phone], [Suppliers].[Fax], [Suppliers].[HomePage] FROM [Su
ppliers] WHERE [Suppliers].[SupplierID] = %s' - PARAMS = (3,); args=(3,)
QUERY = 'SELECT [Suppliers].[SupplierID], [Suppliers].[CompanyName], [Suppliers].[ContactName], [Suppli
ers].[ContactTitle], [Suppliers].[Address], [Suppliers].[City], [Suppliers].[Region], [Suppliers].[Post
alCode], [Suppliers].[Country], [Suppliers].[Phone], [Suppliers].[Fax], [Suppliers].[HomePage] FROM [Su
ppliers] WHERE [Suppliers].[SupplierID] = %s' - PARAMS = (6,); args=(6,)
QUERY = 'SELECT [Suppliers].[SupplierID], [Suppliers].[CompanyName], [Suppliers].[ContactName], [Suppli
ers].[ContactTitle], [Suppliers].[Address], [Suppliers].[City], [Suppliers].[Region], [Suppliers].[Post
alCode], [Suppliers].[Country], [Suppliers].[Phone], [Suppliers].[Fax], [Suppliers].[HomePage] FROM [Su
ppliers] WHERE [Suppliers].[SupplierID] = %s' - PARAMS = (8,); args=(8,)
```

Jak widać zapytania 2,3 I 7 w kolejności są takie same. Ponadto obiekty kategorii i dostawcy, pobierane są w całości, a wyświetlamy tylko ich nazwy. Zatem domyślne zachowanie frameworka nie jest optymalnym rozwiązaniem, gdyż generuje nadmierną ilość zapytań do bazy.

Optymalizacja

Metoda 1

Domyślne zachowanie frameworka Django z panelem Admin za każdym razem pobiera całe obiekty z bazy. A co jeśli chcemy korzystać tylko z jednego pola danej klasy? We wcześniejszym rozdziale omawiano przykład wyświetlenia klienta do wybranego zamówienia. Framework generuje jednak zapytanie o cały obiekt klienta. W tym wypadku nie jest nam to potrzebne, gdyż potrzebujemy z niego tylko pola name. Do klasy OrdersReportAdmin dodano pole list_select_related. Odpowiada ona za pola, które zostaną pobrane używając polecenia JOIN, a nie razem z całym obiektem.

```
class OrdersProxyAdmin(admin.ModelAdmin):
    list_per_page = 10
    list_display = ("orderid", 'customerid', 'orderdate', 'getproducts', 'getcategories', '
getsuppliers')
    list_filter = ("orderdetailsFK__categoryid", "orderdetailsFK__supplierid")
    list_select_related = ('customerid',)
```

Zmiana jaka nastąpiła w generowanych zapytaniach jest przedstawiona poniżej. Zamiast:

```
QUERY = 'SELECT TOP 10 [Orders].[OrderID], [Orders].[CustomerID], [Orders].[EmployeeID], [Orders].[Orde
rDate], [Orders].[RequiredDate], [Orders].[ShippedDate], [Orders].[ShipVia], [Orders].[Freight], [Order
s].[ShipName], [Orders].[ShipAddress], [Orders].[ShipCity], [Orders].[ShipRegion], [Orders].[ShipPostal
Code], [Orders].[ShipCountry] FROM [Orders] ORDER BY [Orders].[OrderID] DESC' - PARAMS = (); args=()
QUERY = 'SELECT [Customers].[CustomerID], [Customers].[CompanyName], [Customers].[ContactName], [Custom
ers].[ContactTitle], [Customers].[Address], [Customers].[City], [Customers].[Region], [Customers].[Post
alCode], [Customers].[Country], [Customers].[Phone], [Customers].[Fax] FROM [Customers] WHERE [Customer
s].[CustomerID] = %s' - PARAMS = ('RATTC',); args=('RATTC',)
```

Otrzymujemy:

```
QUERY = 'SELECT TOP 10 [Orders].[OrderID], [Orders].[CustomerID], [Orders].[EmployeeID], [Orders].[Orde
rDate], [Orders].[RequiredDate], [Orders].[ShippedDate], [Orders].[ShipVia], [Orders].[Freight], [Order
s].[ShipName], [Orders].[ShipAddress], [Orders].[ShipCity], [Orders].[ShipRegion], [Orders].[ShipPostal
Code], [Orders].[ShipCountry], [Customers].[CustomerID], [Customers].[CompanyName], [Customers].[Contac
tName], [Customers].[ContactTitle], [Customers].[Address], [Customers].[City], [Customers].[Region], [C
ustomers].[PostalCode], [Customers].[Country], [Customers].[Phone], [Customers].[Fax] FROM [Orders] LEF
T OUTER JOIN [Customers] ON ([Orders].[CustomerID] = [Customers].[CustomerID]) ORDER BY [Orders].[Order
ID] DESC' - PARAMS = (); args=()
```

Co oznacza, że używając powyższej metody oszczędzamy jedno zapytanie sql dla każdego referujące obiektu.

Naturalnym byłoby użycie identycznego mechanizmu dla pozostałych kluczy obcych łączących tabele, których używamy. Jednak jest to niemożliwe z poziomu tabeli Orders używając atrybutu list_selected_related.

Metoda 2

Kolejnym problemem z wydajnością raportu jest wykonywanie tych samych zapytań do tabeli Products dla każdego dodatkowego pola, którego używamy. Jest to spowodowane użyciem obiektu QuerySet w każdej metodzie.

```
def __init__(self, *args, **kwargs):
        super(Orders, self).__init__(*args, **kwargs)
        self.p = self.orderdetailsFK.all()
```

```
web_1        | (0.007) QUERY = 'SELECT [Products].[ProductID], [Products].[ProductName], [Products].[SupplierID],
[Products].[CategoryID], [Products].[QuantityPerUnit], [Products].[UnitPrice], [Products].[UnitsInStock],
[Products].[UnitsOnOrder], [Products].[ReorderLevel], [Products].[Discontinued] FROM [Products] INNER JOIN [Order
Details] ON ([Products].[ProductID] = [Order Details].[ProductID]) WHERE [Order Details].[OrderID] = %s' - PARAMS =
(12080,); args=(12080,)
web_1        | (0.001) QUERY = 'SELECT [Products].[ProductID], [Products].[ProductName], [Products].[SupplierID],
[Products].[CategoryID], [Products].[QuantityPerUnit], [Products].[UnitPrice], [Products].[UnitsInStock],
[Products].[UnitsOnOrder], [Products].[ReorderLevel], [Products].[Discontinued] FROM [Products] INNER JOIN [Order
Details] ON ([Products].[ProductID] = [Order Details].[ProductID]) WHERE [Order Details].[OrderID] = %s' - PARAMS =
(12080,); args=(12080,)
web_1        | (0.001) QUERY = 'SELECT [Categories].[CategoryID], [Categories].[CategoryName],
[Categories].[Description], [Categories].[Picture] FROM [Categories] WHERE [Categories].[CategoryID] = %s' - PARAMS
= (8,); args=(8,)
web_1        | (0.000) QUERY = 'SELECT [Categories].[CategoryID], [Categories].[CategoryName],
[Categories].[Description], [Categories].[Picture] FROM [Categories] WHERE [Categories].[CategoryID] = %s' - PARAMS
= (8,); args=(8,)
web_1        | (0.000) QUERY = 'SELECT [Categories].[CategoryID], [Categories].[CategoryName],
[Categories].[Description], [Categories].[Picture] FROM [Categories] WHERE [Categories].[CategoryID] = %s' - PARAMS
= (3,); args=(3,)
web_1        | (0.000) QUERY = 'SELECT [Products].[ProductID], [Products].[ProductName], [Products].[SupplierID],
[Products].[CategoryID], [Products].[QuantityPerUnit], [Products].[UnitPrice], [Products].[UnitsInStock],
[Products].[UnitsOnOrder], [Products].[ReorderLevel], [Products].[Discontinued] FROM [Products] INNER JOIN [Order
Details] ON ([Products].[ProductID] = [Order Details].[ProductID]) WHERE [Order Details].[OrderID] = %s' - PARAMS =
(12080,); args=(12080,)
web_1        | (0.001) QUERY = 'SELECT [Suppliers].[SupplierID], [Suppliers].[CompanyName],
[Suppliers].[ContactName], [Suppliers].[ContactTitle], [Suppliers].[Address], [Suppliers].[City],
[Suppliers].[Region], [Suppliers].[PostalCode], [Suppliers].[Country], [Suppliers].[Phone], [Suppliers].[Fax],
[Suppliers].[HomePage] FROM [Suppliers] WHERE [Suppliers].[SupplierID] = %s' - PARAMS = (4,); args=(4,)
web_1        | (0.000) QUERY = 'SELECT [Suppliers].[SupplierID], [Suppliers].[CompanyName],
[Suppliers].[ContactName], [Suppliers].[ContactTitle], [Suppliers].[Address], [Suppliers].[City],
[Suppliers].[Region], [Suppliers].[PostalCode], [Suppliers].[Country], [Suppliers].[Phone], [Suppliers].[Fax],
[Suppliers].[HomePage] FROM [Suppliers] WHERE [Suppliers].[SupplierID] = %s' - PARAMS = (7,); args=(7,)
web_1        | (0.000) QUERY = 'SELECT [Suppliers].[SupplierID], [Suppliers].[CompanyName],
[Suppliers].[ContactName], [Suppliers].[ContactTitle], [Suppliers].[Address], [Suppliers].[City],
[Suppliers].[Region], [Suppliers].[PostalCode], [Suppliers].[Country], [Suppliers].[Phone], [Suppliers].[Fax],
[Suppliers].[HomePage] FROM [Suppliers] WHERE [Suppliers].[SupplierID] = %s' - PARAMS = (8,); args=(8,)
```

Możemy to naprawić dodaniem do raportu konstruktora w którym wykonamy query tylko raz i zapiszemy wynik do tablicy. Pozwala nam to zredukować ilość zapytań o 2 dla każdego zamówienia.

```
web_1        | (0.010) QUERY = 'SELECT [Products].[ProductID], [Products].[ProductName], [Products].[SupplierID],
[Products].[CategoryID], [Products].[QuantityPerUnit], [Products].[UnitPrice], [Products].[UnitsInStock],
[Products].[UnitsOnOrder], [Products].[ReorderLevel], [Products].[Discontinued] FROM [Products] INNER JOIN [Order
Details] ON ([Products].[ProductID] = [Order Details].[ProductID]) WHERE [Order Details].[OrderID] = %s' - PARAMS =
(12080,); args=(12080,)

web_1        | (0.001) QUERY = 'SELECT [Categories].[CategoryID], [Categories].[CategoryName],
[Categories].[Description], [Categories].[Picture] FROM [Categories] WHERE [Categories].[CategoryID] = %s' - PARAMS
= (8,); args=(8,)
web_1        | (0.001) QUERY = 'SELECT [Categories].[CategoryID], [Categories].[CategoryName],
[Categories].[Description], [Categories].[Picture] FROM [Categories] WHERE [Categories].[CategoryID] = %s' - PARAMS
= (8,); args=(8,)
web_1        | (0.000) QUERY = 'SELECT [Categories].[CategoryID], [Categories].[CategoryName],
[Categories].[Description], [Categories].[Picture] FROM [Categories] WHERE [Categories].[CategoryID] = %s' - PARAMS
= (3,); args=(3,)

web_1        | (0.001) QUERY = 'SELECT [Suppliers].[SupplierID], [Suppliers].[CompanyName],
[Suppliers].[ContactName], [Suppliers].[ContactTitle], [Suppliers].[Address], [Suppliers].[City],
[Suppliers].[Region], [Suppliers].[PostalCode], [Suppliers].[Country], [Suppliers].[Phone], [Suppliers].[Fax],
[Suppliers].[HomePage] FROM [Suppliers] WHERE [Suppliers].[SupplierID] = %s' - PARAMS = (4,); args=(4,)
web_1        | (0.001) QUERY = 'SELECT [Suppliers].[SupplierID], [Suppliers].[CompanyName],
[Suppliers].[ContactName], [Suppliers].[ContactTitle], [Suppliers].[Address], [Suppliers].[City],
[Suppliers].[Region], [Suppliers].[PostalCode], [Suppliers].[Country], [Suppliers].[Phone], [Suppliers].[Fax],
[Suppliers].[HomePage] FROM [Suppliers] WHERE [Suppliers].[SupplierID] = %s' - PARAMS = (7,); args=(7,)
web_1        | (0.001) QUERY = 'SELECT [Suppliers].[SupplierID], [Suppliers].[CompanyName],
[Suppliers].[ContactName], [Suppliers].[ContactTitle], [Suppliers].[Address], [Suppliers].[City],
[Suppliers].[Region], [Suppliers].[PostalCode], [Suppliers].[Country], [Suppliers].[Phone], [Suppliers].[Fax],
[Suppliers].[HomePage] FROM [Suppliers] WHERE [Suppliers].[SupplierID] = %s' - PARAMS = (8,); args=(8,)
```

Jak widać zduplikowane zapytania do tabeli Products zniknęły.

Metoda 3

Dotychczas nieznacznie zmniejszyliśmy liczbę zapytań dla każdego zamówienia. Ale wciąż wykonujemy wielokrotne zapytania do tabel Supplier oraz Category. A co jeśli byśmy chcieli mieć tylko jedno zapytanie dla każdego produktu? Nie możemy użyć atrybutu list_select_related, ponieważ odnosi się ono jedynie do pól tabeli Orders. Możemy natomiast spróbować usprawnić zapytanie zdefiniowane w konstruktorze raportu w podobny sposób.

```
def __init__(self, *args, **kwargs):
    super(Orders, self).__init__(*args, **kwargs)
    self.p = self.orderdetailsFK.select_related('categoryid',
'supplierid').all()
```

Jak widać udało nam się. Zapytanie jest wykonywane tylko raz dla każdego produktu. Każdy produkt jest łączony z tabelami Categories oraz Supplier.

```
web_1        | (0.009) QUERY = 'SELECT [Products].[ProductID], [Products].[ProductName],
[Products].[SupplierID], [Products].[CategoryID], [Products].[QuantityPerUnit],
[Products].[UnitPrice], [Products].[UnitsInStock], [Products].[UnitsOnOrder],
[Products].[ReorderLevel], [Products].[Discontinued], [Suppliers].[SupplierID],
[Suppliers].[CompanyName], [Suppliers].[ContactName], [Suppliers].[ContactTitle],
[Suppliers].[Address], [Suppliers].[City], [Suppliers].[Region],
[Suppliers].[PostalCode], [Suppliers].[Country], [Suppliers].[Phone], [Suppliers].[Fax],
[Suppliers].[HomePage], [Categories].[CategoryID], [Categories].[CategoryName],
[Categories].[Description], [Categories].[Picture] FROM [Products] INNER JOIN [Order
Details] ON ([Products].[ProductID] = [Order Details].[ProductID]) LEFT OUTER JOIN
[Suppliers] ON ([Products].[SupplierID] = [Suppliers].[SupplierID]) LEFT OUTER JOIN
[Categories] ON ([Products].[CategoryID] = [Categories].[CategoryID]) WHERE [Order
Details].[OrderID] = %s' - PARAMS = (12080,); args=(12080,)
```

Metoda 3

Już niewiele można ulepszyć w tym zapytaniu. Jedną z tych rzeczy są niepotrzebnie pobierane pola tabel, z których używamy tylko niektórych wartości. Można tego uniknąć w łatwy sposób używając selektora only, który pozwala nam zdefiniować listę pól jakie chcemy dostać z bazy.

```
def __init__(self, *args, **kwargs):
    super(Orders, self).__init__(*args, **kwargs)
    self.p = self.orderdetailsFK.select_related('categoryid',
'supplierid').only('supplierid', 'supplierid__companyname', 'categoryid',
'categoryid__categoryname', 'productname').all()
```

Jak widac pobierane sa tylko uzywane pola ze wszystkich tabel.

```
web_1        | (0.001) QUERY = 'SELECT [Products].[ProductID], [Products].[ProductName],
[Products].[SupplierID], [Products].[CategoryID], [Suppliers].[SupplierID],
[Suppliers].[CompanyName], [Categories].[CategoryID], [Categories].[CategoryName] FROM
[Products] INNER JOIN [Order Details] ON ([Products].[ProductID] = [Order
Details].[ProductID]) LEFT OUTER JOIN [Suppliers] ON ([Products].[SupplierID] =
[Suppliers].[SupplierID]) LEFT OUTER JOIN [Categories] ON ([Products].[CategoryID] =
[Categories].[CategoryID]) WHERE [Order Details].[OrderID] = %s' - PARAMS = (11075,);
args=(11075,)
```

Metoda 4.

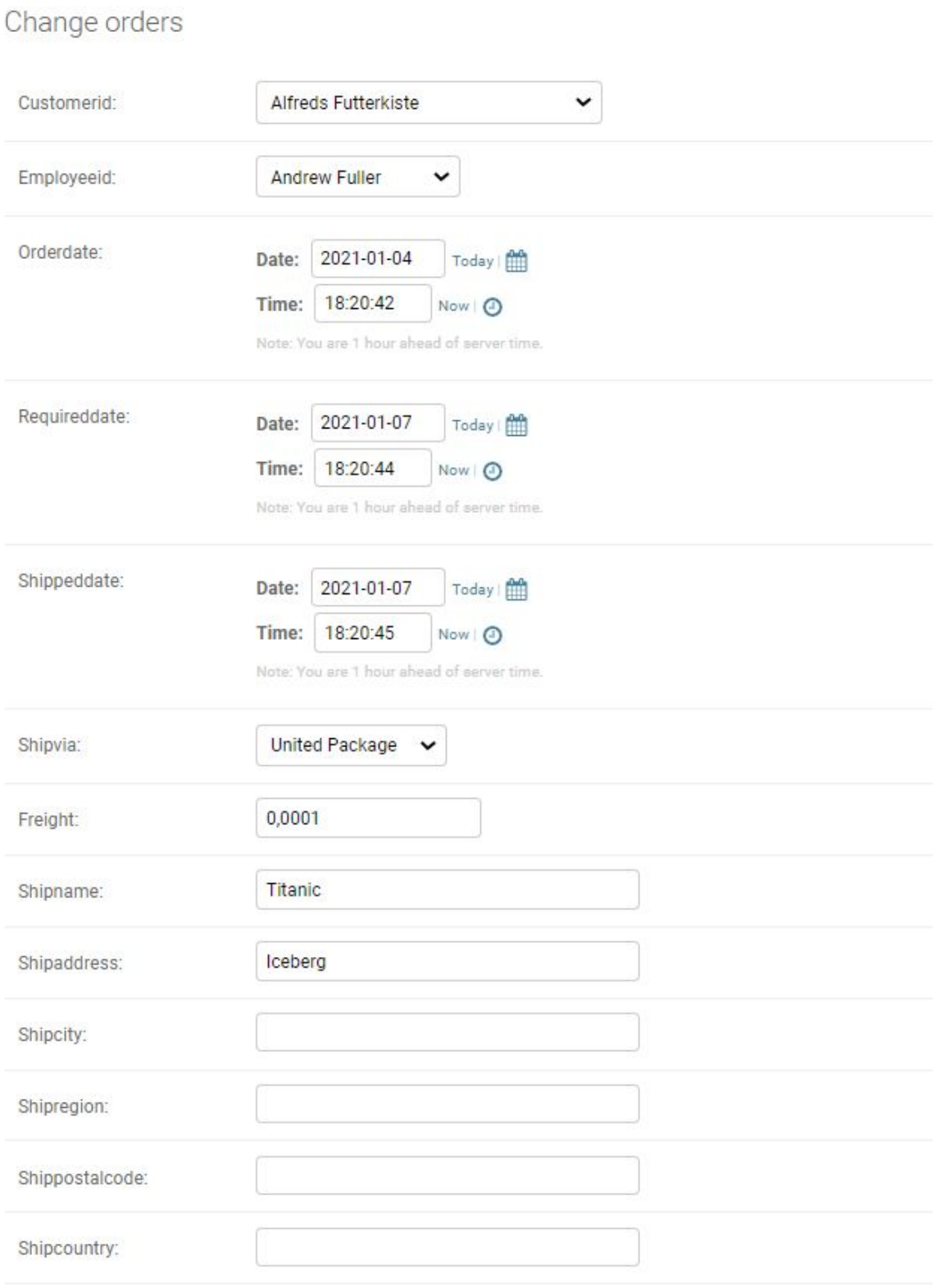
Po wielu godzinach spedzonych w dokumentacji Django udalo nam sie sprawic, ze beda tylko dwa zapytania dla calego widoku. Wymagalo to przepisania znacznej czesci kodu odpowiedzialnego za raport, ale oplacilo sie.

```
class OrdersProxy(Orders):
    class Meta:
        verbose_name_plural = 'Orders Reports'
        proxy = True

class OrdersProxyAdmin(admin.ModelAdmin):
    def get_queryset(self, request):
        pref = Prefetch('orderdetailsFK', \
            Products.objects.select_related('categoryid', 'supplierid') \
                .only('productid', 'productname', 'categoryid',
'categoryid__categoryname', 'supplierid', 'supplierid__companyname'), to_attr='prod')
        return super(OrdersProxyAdmin, self).get_queryset(request) \
            .select_related('customerid') \
            .prefetch_related(pref) \
            .only('orderid', 'orderdate', 'customerid', 'customerid__companyname',
'orderdetailsFK__productid', 'orderdetailsFK__productname', \
                'orderdetailsFK__supplierid', 'orderdetailsFK__supplierid__companyname',
'orderdetailsFK__categoryid',
'orderdetailsFK__categoryid__categoryname')

    def getorderid(self, obj):
        return obj.orderid

    def getcustomerid(self, obj):
        return obj.customerid

    def getorderdate(self, obj):
        return obj.orderdate

    def getproducts(self, obj):
        return ", ".join([p.productname for p in obj.prod])

    def getcategories(self, obj):
        return ", ".join([c for c in set([p.categoryid.categoryname for p in obj.prod])])

    def getsuppliers(self, obj):
        return ", ".join([s for s in set([p.supplierid.companyname for p in obj.prod])])

    list_per_page = 10
    list_display = ('getorderid', 'getcustomerid', 'getorderdate', 'getproducts',
'getcategories', 'getsuppliers')
    list_filter = ("orderdetailsFK__categoryid", "orderdetailsFK__supplierid")
```

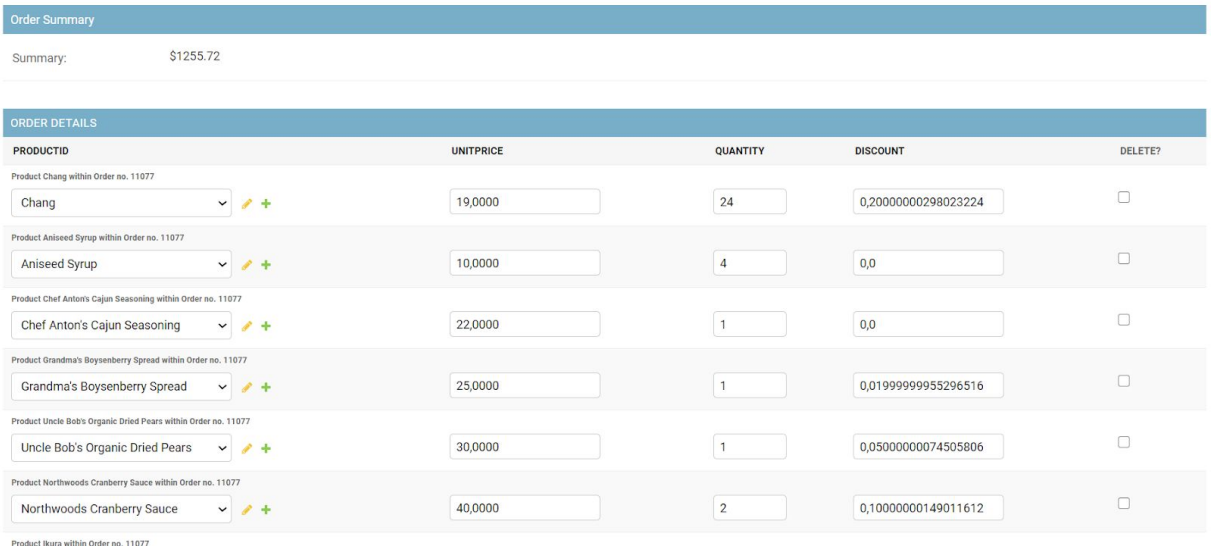
```
web_1        | (0.006) QUERY = 'SELECT TOP 10 [Orders].[OrderID], [Orders].[CustomerID],
[Orders].[OrderDate], [Customers].[CustomerID], [Customers].[CompanyName] FROM [Orders]
LEFT OUTER JOIN [Customers] ON ([Orders].[CustomerID] = [Customers].[CustomerID]) ORDER
BY [Orders].[OrderID] DESC' - PARAMS = (); args=()
web_1        | (0.011) QUERY = 'SELECT ([Order Details].[OrderID]) AS
[_prefetch_related_val_orderid_id], [Products].[ProductID], [Products].[ProductName],
[Products].[SupplierID], [Products].[CategoryID], [Suppliers].[SupplierID],
[Suppliers].[CompanyName], [Categories].[CategoryID], [Categories].[CategoryName] FROM
[Products] INNER JOIN [Order Details] ON ([Products].[ProductID] = [Order
Details].[ProductID]) LEFT OUTER JOIN [Suppliers] ON ([Products].[SupplierID] =
[Suppliers].[SupplierID]) LEFT OUTER JOIN [Categories] ON ([Products].[CategoryID] =
[Categories].[CategoryID]) WHERE [Order Details].[OrderID] IN (%s, %s, %s, %s, %s, %s,
%s, %s, %s, %s)' - PARAMS = (12080, 12079, 11079, 11077, 11076, 11075, 11074, 11073,
11072, 11071); args=(12080, 12079, 11079, 11077, 11076, 11075, 11074, 11073, 11072,
11071)
```

Jak widac pierwsze zapytanie pobiera liste orderow, a drugie ich pola.

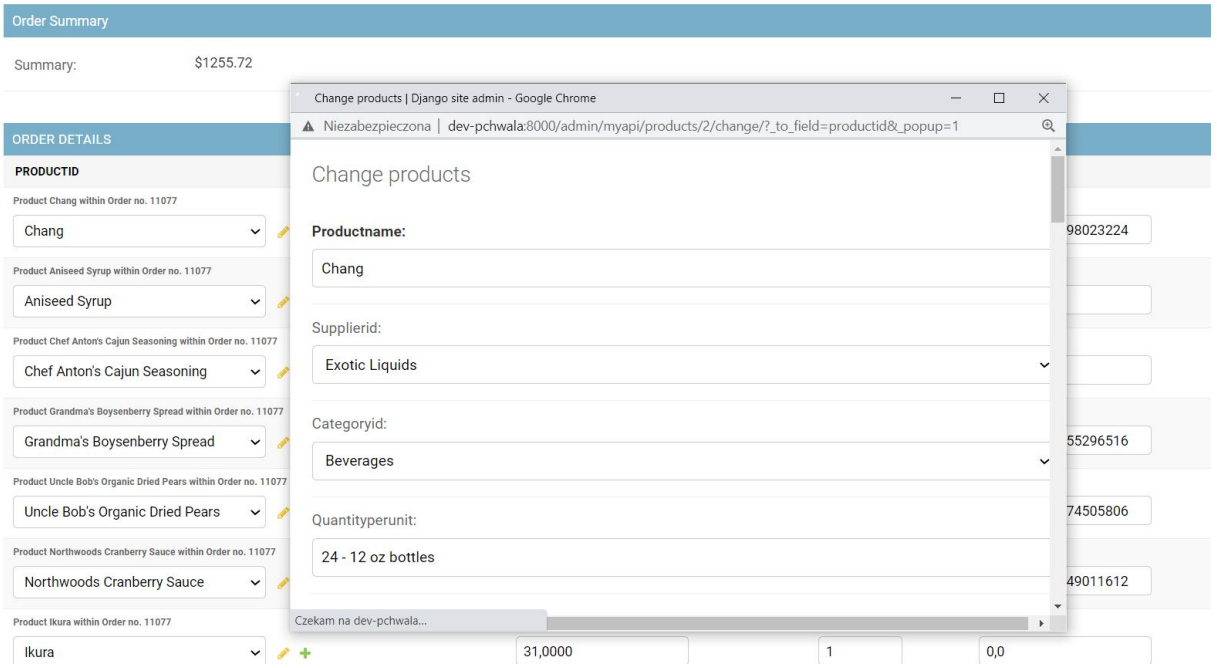
Galeria


Change orders
Customerid:
Alfreds Futterkiste
Employeeid:
Andrew Fuller
Orderdate:
Date: 2021-01-04 Today
Time: 18:20:42 Now
Note: You are 1 hour ahead of server time.
Requireddate:
Date: 2021-01-07 Today
Time: 18:20:44 Now
Note: You are 1 hour ahead of server time.
Shippeddate:
Date: 2021-01-07 Today
Time: 18:20:45 Now
Note: You are 1 hour ahead of server time.
Shipvia:
United Package
Freight:
0,0001
Shipname:
Titanic
Shipaddress:
Iceberg
Shipcity:
Shipregion:
Shippostalcode:
Shipcountry:

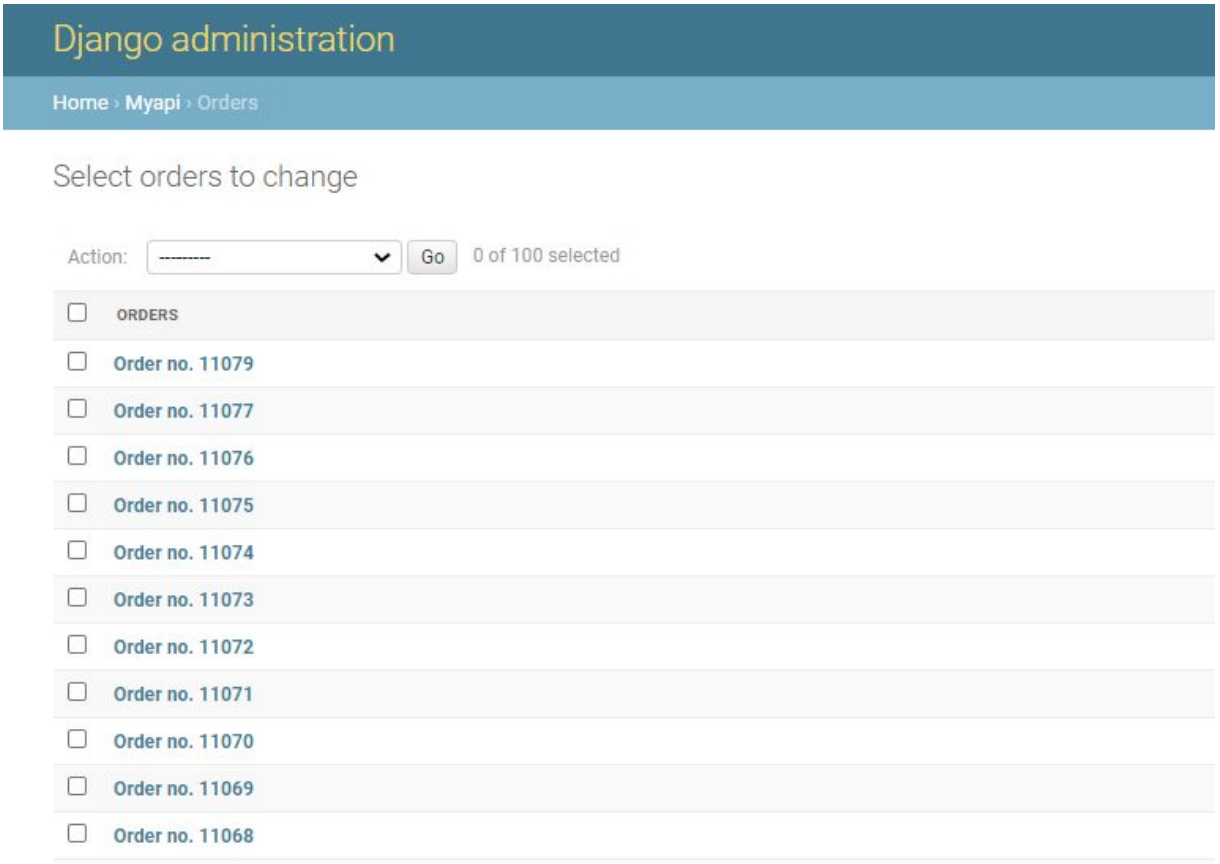
Dodanie/edycja zamówienia.



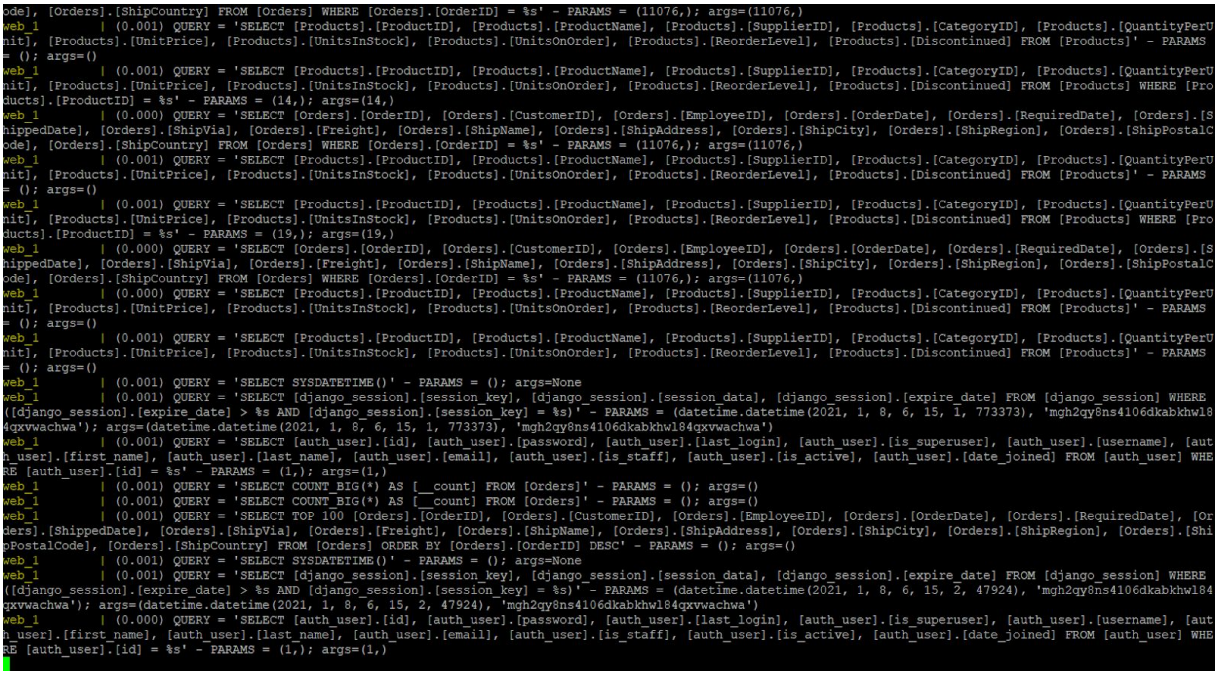
Podsumowanie i widok produktów należących do zamówienia.



Edycja produktu z poziomu zamówienia.



Lista zamówień.



Podgląd na żywo wykonywanych zapytań do bazy danych.

Wnioski

wnioski