Jacek Wołoszyn

Modyfikacja modelu aplikacji realizującej transakcje giełdowe na podstawie rzeczywistego strumienia danych o wartość dywidend

Dydaktyka Informatyki 12, 209-217

2017

Dydaktyka Informatyki 12(2017)
ISSN 2083-3156
DOI: 10.15584/di.2017.12.25
http://www.di.univ.rzeszow.pl

Wydział
Matematyczno-Przyrodniczy UR
Laboratorium Zagadnień
Społeczeństwa Informacyjnego

Jacek WOŁOSZYN

Dr inż., Uniwersytet Technologiczno-Humanistyczny w Radomiu, Wydział Informatyki i Matematyki, Katedra Informatyki, ul. Malczewskiego 29, 26-600 Radom; jacek@delta.pl

MODYFIKACJA MODELU APLIKACJI REALIZUJĄCEJ TRANSAKCJE GIEŁDOWE NA PODSTAWIE RZECZYWISTEGO STRUMIENIA DANYCH O WARTOŚĆ DYWIDEND

MODIFICATION OF THE APPLICATION MODEL REALIZES EXCHANGE TRANSACTIONS BASED ON ACTUAL DATA STREAM BY THE VALUE OF DIVIDENDS

Słowa kluczowe: baza danych, giełda, dywidendy, model.
Keywords: databases, stock exchange, dividend, model.

Streszczenie

W poniższym artykule opisano rozwiązanie problemu uwzględnienia wartości portfela użytkownika o wartość dywidendy dla transakcji opartych na rzeczywistym strumieniu danych.

Summary

The following article describes a solution to take into account the value of the portfolio's value by the dividend for transactions based on the actual data stream.

Wstęp

Opisane zagadnienie polega na modyfikacji działającego już systemu informatycznego, w którym zarejestrowani użytkownicy posiadający wymagany poziom uprawnień realizują transakcje giełdowe polegające na zakupie i sprzedaży akcji na parkiecie GPW w oparciu o rzeczywisty strumień danych. Zastana sytuacja działa poprawnie zwiększając lub zmniejszając wartości portfeli użytkowników. Problem polega na tym, że w pobieranych zasobach strumienia informacji nie są uwzględniane wartości wynikające z wypłat dywidend.

Stan obecny sytuacji

System transakcji giełdowych jest oparty na klasycznej komunikacji typu klient – serwer. Całość systemu działa w rozproszonym środowisku sieciowym opartym na protokole TCP/IP. Do serwera wysyłają swoje żądania klienci, a ten je realizuje w czasie rzeczywistym w oparciu o zadany algorytm uwzględniając informacje napływające w strumieniu danych z serwera giełdowego. Wszystkie dane po odpowiednim przetworzeniu są przechowywane w bazie danych[1].

System giełdowy posiada wiele modeli w swojej strukturze pozwalających osiągnąć zamierzony efekt. Jednym z nich jest struktura opisująca użytkownika pozwalająca na stworzenie jego charakterystyki /profilu/, a także zarządzaniu nim w procesie autoryzacji.

Model opisujący użytkownika w systemie.

```
user = models.OneToOneField(User)
name = models.CharField(max_length=25, verbose_name="Imię")
lastname = models.CharField(max_length=40, verbose_name="Nazwisko")
email = models.EmailField(verbose_name="Email")
gender = models.CharField(max_length=1, verbose_name="Płeć")
birthdate = models.DateField(verbose_name="Data urodzenia*")
age = models.CharField(max_length=6, verbose_name="Wiek")
city = models.CharField(max_length=50, verbose_name="Miejsce zamieszkania")
……
```

Rys. 1. Fragment modelu opisującego strukturę użytkownika w systemie

Użytkownik jako podstawowa jednostka w systemie powiązana jest za pomocą relacji w tabelach bazy danych z innymi strukturami systemu i pozwala na jednoznaczną identyfikację chociażby portfeli, czy dokonywanych transakcji, jak i sposobu prezentowania informacji.

Model portfela użytkownika definiowany jest w następujący sposób:

```
user = models.ForeignKey(User, verbose_name="Właściciel")
name = models.CharField(max_length=50, verbose_name="Nazwa)
tagi = models.CharField(max_length=100, verbose_name="Tagi")
curr = models.CharField(max_length=6, verbose_name="Waluta")
…...
```

Rys. 2. Fragment struktury modelu opisujący portfel użytkownika

[1] M. Goodrich, R. Tamassia, M. Goldwasser, *Data Structures and Algorithms in Python*, Wiley 2013.

Pole user jednoznacznie identyfikuje właściwego użytkownika portfela i tym samym podczas procesu agregacji[2] danych uwzględnia tylko właściwe rekordy należące do użytkownika portfela.

Jednym z istotnych pól omawianych jest pole vbud definiujące wartość gotówki posiadanej w portfelu. Jest to pole typu FloatField i jest obliczane jako suma gotówki wniesionej do systemu, jak i suma wszystkich wykonanych transakcji kupna sprzedaży, jak i poniesionych przy tym kosztów związanych z realizacją transakcji jak i możliwością bycia aktywnym użytkownikiem systemu.

Opis problemu

Przykładowy wycinek strumienia danych docierającego do systemu podlegającemu dalszej obróbce przez algorytm wygląda następująco:

```
INSERT INTO opozn.transakcje (id, time_t, isin, nazwa, data, czas, num_tr, kurs, wolumen, wartosc, grupa, kateg, mic) VALUES (90169528, 1482509099, 'PLMSTSD00019', 'POLIMEXMS', '2016-12-23', '17:04:59', 570, 3.67, 125, 458.75, '03', 'A', 'XWAR');
INSERT INTO opozn.transakcje (id, time_t, isin, nazwa, data, czas, num_tr, kurs, wolumen, wartosc, grupa, kateg, mic) VALUES (90169527, 1482509099, 'PLMSTSD00019', 'POLIMEXMS', '2016-12-23', '17:04:59', 569, 3.67, 350, 1284.5, '03', 'A', 'XWAR');
INSERT INTO opozn.transakcje (id, time_t, isin, nazwa, data, czas, num_tr, kurs, wolumen, wartosc, grupa, kateg, mic) VALUES (90169526, 1482509099, 'PLMSTSD00019', 'POLIMEXMS', '2016-12-23', '17:04:59', 568, 3.67, 41, 150.47, '03', 'A', 'XWAR');
INSERT INTO opozn.transakcje (id, time_t, isin, nazwa, data, czas, num_tr, kurs, wolumen, wartosc, grupa, kateg, mic) VALUES (90169525, 1482509099, 'PLMSTSD00019', 'POLIMEXMS', '2016-12-23', '17:04:59', 567, 3.67, 350, 1284.5, '03', 'A', 'XWAR');
INSERT INTO opozn.transakcje (id, time_t, isin, nazwa, data, czas, num_tr, kurs, wolumen, wartosc, grupa, kateg, mic) VALUES (90169524, 1482509099, 'PLMSTSD00019', 'POLIMEXMS', '2016-12-23', '17:04:59', 566, 3.67, 134, 491.78, '03', 'A', 'XWAR');
```

Po wstępnej analizie zawartości strumienia informacji można zauważyć, że pola zawierają numer transakcji, czas transakcji, nazwe ISIN instrumentu, nazwę skróconą, datę, czas, kurs, wolumen, typ…, jednak nie ma żadnych informacji dotyczących dywidend.

[2] A. Downey, *Python for Software Design*, Cambridge University Press 2009; A. Downey, *Think Python*, O'Reilly 2012.

Rozwiązanie problemu

Powstaje zatem pytanie, jak takie informacje uwzględnić. Skoro nie ma takich informacji w strumieniu danych należy je uwzględnić ze źródeł zewnętrznych i odpowiednio zsynchronizować. Właściwe drogi rozwiązania problemu to zaciągnięcie danych z zewnętrznej bazy danych, czytanie z zewnętrznego pliku, wpisywanie danych przez administratora systemu do specjalnie utworzonej bazy i uwzględnianie jej w algorytmie. Pierwsze rozwiązanie nie jest możliwe do realizacji, ponieważ takie dane nie są strumieniowane, pozostają dwie możliwości. Rozwiązanie z zaczytywaniem informacji z pliku zewnętrznego może działać bez problemu, lecz wymagałoby dodatkowych algorytmów sprawdzających poprawność struktury pliku, jak i zabezpieczenia przed przypadkowym podwójnym wczytaniem tych samych informacji. Pozostało więc rozwiązanie trzecie polegające na utworzeniu dodatkowej tabeli w systemie i umieszczaniu informacji pod nadzorem administratora poprzez specjalnie dedykowany do tego formularz.

Dywidendy

Dywidenda do część zysku netto przeznaczona dla właściciela akcji.

Tabela powinna zatem zawierać pola, które powinny jednoznacznie identyfikować instrument, wartość dywidendy na akcję, jak również datę nabycia z prawem dywidend, datę prawa do dywidendy, jak i datę wypłaty dywidendy. Administrator wpisuje do systemu pola zawierające nazwę akcji, wartości dywidendy, datę prawa do dywidendy, datę wypłaty dywidendy. Natomiast pole data nabycia z prawem do dywidend jest kalkulowana po zakończeniu sesji w tym dniu pojawia się w należnościach.

Po uwzględnieniu tych informacji została zdefiniowana tabela umożliwiająca przechowywanie takich informacji:

```
CREATE TABLE `dywidendy` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `wartosc` double NOT NULL,
  `data_na` date DEFAULT NULL,
  `data_pr` date NOT NULL,
  `data_wy` date NOT NULL,
  `naz` varchar(50) COLLATE utf8_polish_ci,
  `status` varchar(10) COLLATE utf8_polish_ci NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_polish_ci
```

Rys. 3. Definicja tabeli zawierającej dywidendy

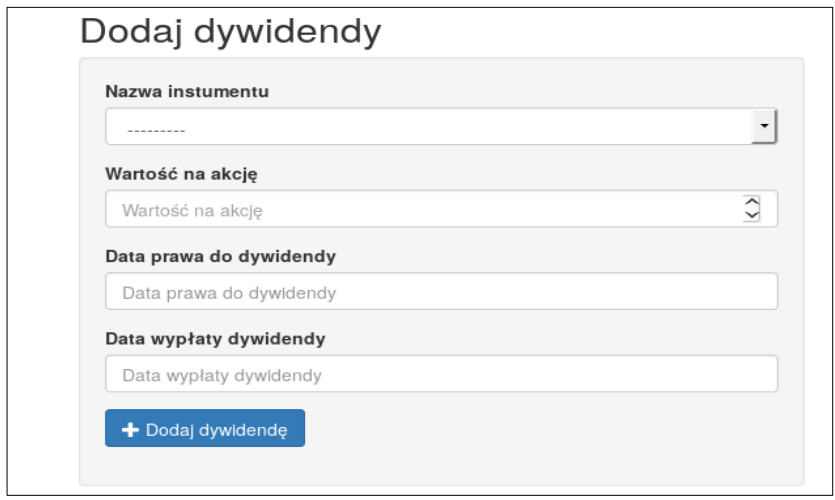
Jak można łatwo zauważyć, pojawiło się też dodatkowe pole ‘status’, które będzie wykorzystywane w algorytmie do selekcji już uwzględnionych dywidend.

Etapem kolejnym jest umożliwienie osobie odpowiedzialnej za umieszczanie dywidend w systemie. Będzie się to odbywało za pomocą formularza dostępnego tylko dla uprawnionej osoby.

```
def add_dyw(request):

    result = []
    data_list = PortVal.objects.all()
    data_l = data_list.values('datav')
    for i in data_l:
        if i['datav'] not in result:
            result.append(i['datav'])
    v = (max(result))
    dl = len(result)
    vd = v – timedelta(days=2)

    if request.method == 'POST':
        form = Add_DywidendyForm(request.POST)

        if form.is_valid():
            dyw = form.save(commit=False)
            dn = dyw.data_pr - timedelta(days=2)
            if dn in result:
                dyw.data_na = dn
            else:
                dyw.data_na = result[dl-1]
            dyw.save()
            return HttpResponseRedirect('/to_page/')

    else:
        user = request.user
        profile = user.profile
        form = Add_DywidendyForm(instance=profile)

    args = {}
    args.update(csrf(request))
    args['form'] = form

    return render_to_response('to_page.html', args)
```

Rys. 4. Listing procedury generującej formularz

Na powyższym listingu w pierwszej kolejności jest obliczana data nabycia z prawem do dywidend. W procedurze tej uwzględniane są daty, kiedy notowania miały miejsce w dni robocze. W kolejnych wierszach generowany jest formularz, w którego pola będą wpisywane niezbędne informacje do obliczania dywidend. Pole 'status', o którym była mowa, może przyjąć dwie wartości. Domyślnie jest nadawana wartość 'ready', która będzie informowała, że dywidenda nie została jeszcze uwzględniona, natomiast w kroku kolejnym będzie przyjmowała wartość 'ok', która będzie odrzucana przez filtr[3] i tym samym będzie uniemożliwiała powtórne uwzględnienie jej w obliczeniach.



Rys. 5. Formularz pozwalający na umieszczanie danych w bazie

Dla prawidłowego funkcjonowania pod pole 'nazwa instrumentu' została podpięta lista dostępnych instrumentów.

Po wypełnieniu formularza i poprawnym zapisie /została zastosowana walidacja sprawdzająca poprawność danych przed zapisem/ dane zostają umieszczone w tabeli bazy danych.

Za pomocą zapytania SQL można sprawdzić zawartość tabeli dywidendy:

```
SELECT * FROM dywidendy;

id    war    data_na       data_pr  data_wy naz          status
1     0.4    2016-10-25    2016-11-19      2016-11-22     11BIT   ready
```

Rys. 6. Listing zawartość tabeli dywidendy

[3] Y. Hilpisch, *Derivatives Analytics with Python*, Wiley 2015.

Rezultatem dotychczas opisywanych działań jest zapis rekordu w tabeli dywidendy, które zawiera wszystkie niezbędne informacje oraz dodatkowo pole status z domyślną wartością ‘ready’ informującą o tym, że dywidenda nie została jeszcze użyta. W testowej bazie danych ostatnie notowanie było zapisane w dniu 25-10-2016 r., co oznacza ze zastosowany algorytm działa poprawnie.

Jednak, do tej pory w systemie istnieją tylko zapisy o dywidendach, co teraz należy zrobić, aby te dywidendy wyliczyć?

Obliczenie wartości dywidend

Aby ostatecznie zwiększyć wartość gotówki w portfelu, należy uwzględnić wpisane dywidendy i sprawdzić, czy w poszczególnych portfelach użytkowników znajdują się akcje, dla których generowane są dywidendy, a jeśli tak to obliczyć wartość dywidendy jako iloczyn posiadanych akcji uprawnionych go otrzymania dywidendy oraz wartości przypadającej na każdą z nich i zapisać te informacje w tabeli.

Do realizacji tego zadania utworzono kolejną tabelę dywtb, w której takie informacje będą gromadzone:

```
CREATE TABLE `dywtb` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `instrd` varchar(50) COLLATE utf8_polish_ci NOT NULL,
  `iloscd` double NOT NULL,
  `nalezd` double NOT NULL,
  `gotowd` double NOT NULL,
  `datawd` date NOT NULL,
  `statud` varchar(10) COLLATE utf8_polish_ci NOT NULL,
  `portd` varchar(50) COLLATE utf8_polish_ci NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_polish_ci
```

Rys. 7. Tworzenie tabeli dywtb

W tabeli znajdują się tylko niezbędne informacje, pozwalające obliczyć wartość dywidendy: w tym instrd nazwa instrumentu, iloscd ilość akcji, należność – aktywna przed datą wypłaty i gotówd – przepisywana wartość z nalezd w dniu wypłaty dywidendy, a także przynależność do portfela, a tym samym i użytkownika portd.

Procedura realizująca ten zapis jest zamieszczona na kolejnej stronie.

```
def dywid(request):

    dyw = Dywidendy.objects.filter(status='ready').values('id', 'naz', 'wartosc', 'data_na',
'data_pr', 'data_wy', 'status')

    for i in dyw:
        pdet = PortDet.objects.filter(
            instr__nazwa=i['naz'], datan__lte=i['data_na'], dataz__lte=i['data_na']).values(
            'port__name', 'ilosc', 'instr__nazwa')
        pd = pdet.values('port__name').annotate(suma=(Sum(F('ilosc'))))
        for j in pd:
            DywTb.objects.create(portd=j['port__name'], instrd=i['naz'], iloscd=j['suma'],
                                 nalezd=j['suma'] * i['wartosc'], gotowd=0,
                                 datawd=i['data_wy'])
        Dywidendy.objects.filter(id=i['id']).update(status='ok')

    return render_to_response('to_page1html')
```

W pierwszym kroku wyszukiwane są wszystkie dywidendy wpisane do tabeli zawierające status ‘ready’, czyli nie zostały jeszcze nigdy użyte.

Z kolei sprawdzane są wszystkie transakcje w bazie zawierające nazwę instrumentu, jak dywidendy, jak również dodatkowe kryteria dat.

Jeśli kryteria zostaną spełnione wówczas zostaje wygenerowany rekord i zapisany w bazie dywtb z naliczonymi dywidendami, a raz użyta przyjmuje status ‘ok’, co oznacza, że nie może już być nigdy używana powtórnie.

id	instrd	iloscd	nalezd	gotd	datawd	std	portd
1	11BIT	438	175.2	0	2016-11-22	todo	Portfel aaa
2	11BIT	7	2.8	0	2016-11-22	todo	Portfel gosia

Rys. 8. Wynik działania procedury dywid

Wynikiem działania procedury są zapisy rekordów do bazy danych, o ile oczywiście są spełnione warunki.

W tym przypadku zostały zaliczone wartości 175,20 zł dla portfela o nazwie aaa oraz 2,80 zł dla portfela o nazwie gosia.

Kolejnym, już ostatnim, etapem jest uwzględnienie tych wartości w całości gotówki w portfelu, co wiąże się z tym, że przy zamknięciu dnia dla portfeli do wartości vbud naliczonej dla portfela należy dodać wyliczone wartości z pola gotowd z tabeli dywtb dla poszczególnych portfeli.

Wnioski

Przedstawiona metoda jest tylko jednym z możliwych sposobów podejścia do problemu. Zdecydowanie najkorzystniejszym rozwiązaniem byłoby, aby w przekazywanym strumieniu danych były zawierane informacje o wartościach dywidend. To pozwoliłoby na automatyczne bezobsługowe napisanie odpowiednich procedur uwzględniających wartość portfela dywidendy. Rozwiązanie takie jest możliwe jednak tylko w przypadku udziału strony trzeciej.

Zastosowanie natomiast procedury opisanej w tym artykule pozwala rozwiązać ten problem, nie obciążając istotnie zasobów systemowych, jak i osoby odpowiedzialnej za zamieszczanie dywidend w bazie, gdyż sytuacje takie nie zdarzają się zbyt często.

Literatura

Downey A., *Python for Software Design*, Cambridge University Press 2009.
Downey A., *Think Python*, O'Reilly 2012.
Goodrich M., Tamassia R., Goldwasser M., *Data Structures and Algorithms in Python*, Wiley 2013.
Hellman D., *The Python Standard Library by Example*, Addison-Wesley 2011.
Hilpisch Y., *Derivatives Analytics with Python*, Wiley 2015.
Payne J., *Beginning Python*, Wrox 2010.
Summerfield M., *Programming in Python 3*, Addison-Wesley 2010.
Ziade T., *Packt, Expert Python Programming*, Publishing 2008.