JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 11 (2019) 82-85
Wysłane: 2018-11-30
Przyjęte: 2018-12-01

Wydajność pracy z bazami danych aplikacji tworzonych w Django

Bartosz Nejman*, Beata Pańczyk
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Tworzenie aplikacji internetowych składa się z wielu procesów. Jednym z nich jest wybór odpowiedniego systemu bazy danych gdyż to od niej może w dużej mierze zależeć szybkość działania aplikacji. Duża liczba dostępnych systemów baz danych sprawia iż wybór ten nie jest łatwy. Niniejszy artykuł przedstawia porównanie wydajności pracy z bazami danych na przykładzie aplikacji tworzonych w Django oraz trzech systemów bazodanowych: MySQL, PostgreSQL oraz MongoDB.

Słowa kluczowe: Django; MongoDB; MySQL; PostgreSQL

*Autor do korespondencji.

Adres e-mail: nejmanbartosz@gmail.com

Efficiency of databases in Django-based applications

Bartosz Nejman*, Beata Pańczyk
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. Development of web applications consists of many processes. One of them is choosing appropriate database management system which may have huge impact on application performance. Large availability of database management systems makes it not an easy choice. The goal of this paper is to compare efficiency of databases in Django-based applications and three different database management systems: MySQL, PostgreSQL and MongoDB.

Keywords: Django; MongoDB; MySQL; PostgreSQL

*Corresponding author.

E-mail address: nejmanbartosz@gmail.com

1. Wstęp

Na przestrzeni ostatnich lat można dostrzec znaczny postęp w tworzeniu aplikacji internetowych. Jeszcze kilka lat temu strony WWW składały się głównie ze statycznego kodu HTML, przez co nie można było nadać im miana aplikacji. Wraz z postępującym rozwojem aplikacji internetowych zaczęto przechowywać co raz więcej danych. Treści na stronie w większości generowane są dynamicznie, na przykład poprzez pobranie ich z bazy danych . Wyższy poziom zaawansowania aplikacji wiążę się jednak z dłuższym czasem ładowania strony co jest dużym problemem w obecnych czasach. Rozwiązaniem jest dobranie odpowiednich narzędzi, a jednym z najważniejszych jest baza danych . Dobranie odpowiedniego systemu dla konkretnej aplikacji nie jest łatwe. Przeprowadzenie badań wydajności poszczególnych systemów bazodanowych może być bardzo kosztowne. Z pomocą przychodzą narzędzia wykorzystujące mapowanie obiektowo-relacyjne (ang. Object-Relational Mapping – ORM). Zaletą tego typu narzędzi jest możliwość przełączenia się między różnymi systemami bazodanowymi bez konieczności przebudowywania całej aplikacji .

2. Cel i teza

Celem niniejszego artykułu jest bezpośrednie porównanie wydajności różnych systemów bazodanowych, wykorzystując aplikację webową napisaną w Django. Do realizacji badań zostaną wykorzystane najpopularniejsze systemy bazodanowe:

- relacyjny MySQL,
- relacyjnyPostgreSQL,
- nierelacyjny system MongoDB .

Za tezę badawczą przyjęto:

Dla aplikacji tworzonych w Django, relacyjne bazy danych (MySQL i PostgreSQL) są bardziej wydajne w porównaniu do nierelacyjnej bazy MongoDB.

3. Systemy zarządzania bazami danych

3.1. MongoDB

MongoDB to nierelacyjny system bazodanowy napisany w języku C++. Wysoką wydajność w porównaniu do relacyjnych systemów zapewnia sposób przechowywania danych w postaci dokumentów przypominających format JSON. Tego typu sposób zapewnia kompatybilność z większością najpopularniejszych języków programowania między innymi Python, czy PHP .

Najważniejsze cechy MongoDB [6]:

1) dane przechowywane są w kolekcjach w postaci dokumentów w formacie BSON (binary JSON);

2) zaawansowany język zapytań wspierający operacje wejścia/wyjścia i agregację danych;
3) wykorzystywanie indeksów pozwala na szybsze wykonywanie zapytań;
4) automatyczna obsługa sytuacji awaryjnych (ang. failover);
5) pozioma skalowalność zapewniona dzięki dystrybuowaniu danych po klastrach podzespołów (w j.ang sharding) .

3.2. MySQL

MySQL jest jednym z najpopularniejszych systemów zarządzania relacyjnymi bazami danych (ang. Relational Database Management System - RDBMS). Dzięki sprawdzonej wydajności, niezawodności i łatwości użycia MySQL został najczęściej wybieranym systemem bazodanowym w aplikacjach webowych. Wykorzystywany jest na całym świecie, między innymi przez takich gigantów jak Facebook, Twitter, czy Youtube . Historia MySQL sięga roku 1995 kiedy to pojawiło się pierwsze wydanie. MySQL jest dostępny na licencji open source jednak właścicielem jest Oracle, które również udostępnia licencje komercyjne. Na rynku główną konkurencję stanowią PostgreSQL, Oracle oraz Microsoft SQL Server. Na ich tle MySQL wyróżnia się przede wszystkim następującymi cechami :

1) wydajność - myślą przewodnią projektu była szybkość działania;
2) łatwość użycia – podstawowa konfiguracja sprowadza się do kilku kliknięć;
3) niskie koszty użytkowania dzięki wsparciu ogromnej społeczności.

3.3. PostgreSQL

PostgreSQL znany również jako Postgres, to obok MySQL jeden z najpopularniejszych systemów open source do zarządzania obiektowo-relacyjnymi bazami danych . PostgreSQL zyskał silną reputację dzięki sprawdzonej architekturze, niezawodności, integracji danych oraz wysokiej skalowalności. Wspiera wszystkie główne systemy operacyjne. Od 2001 roku jest również zgodny ze standardem ACID zapewniając niepodzielność, spójność, izolację transakcji oraz trwałość danych.

4. Porównanie systemów

4.1. Środowisko testowe

Do przeprowadzenia badań została napisana aplikacja webowa w języku Python w wersji 3.7.0 z wykorzystaniem o frameworka Django w wersji 2.1.1. Dla każdego systemu bazy danych została wykorzystana ta sama aplikacja testowa.

Wszystkie testy zostały przeprowadzone na tym samym sprzęcie: Lenovo y510P, 64 bitowy system Windows 10 PRO. Komputer został wyposażony w procesor Intel® Core™ i5-4200 M o taktowaniu 2.5GHz, kartę graficzną NVIDIA GeForce GT 755M, 8 GB pamięci ram DDR3 o częstotliwości 1600 MHz oraz dysk SSD WD GREEN o pojemności 240GB. Wykorzystane bazy danych to MySQL 5.6.37, PostgreSQL 10.6 oraz MongoDB 3.4.7.

4.2. Scenariusze testowe

Scenariusze testowe realizują fundamentalne operacje wykonywane w aplikacjach webowych. Te operacje to wstawianie, odczytywanie danych oraz aktualizowanie danych. Testy są wykonywane na tabelach prostych i tabelach złożonych, które posiadają odwołania do innych tabel. Dla każdego scenariusza i każdej bazy danych, operacje zostały wykonane 1000 razy, a ich czas zsumowany. Tabela 1 przedstawia zestawienie scenariuszy.

Tabela . Scenariusze testów

Numer scenariusza	Opis scenariusza
1	Utworzenie jednego modelu bez relacji
2	Utworzenie jednego modelu z relacjami
3	Utworzenie stu modeli bez relacji
4	Utworzenie stu modeli z relacjami
5	Pobranie jednego modelu bez relacji
6	Pobranie jednego modelu z relacjami
7	Pobranie stu modeli bez relacji
8	Pobranie stu modeli z relacjami
9	Warunkowe pobranie modeli bez relacji
10	Warunkowe pobranie modeli z relacjami
11	Warunkowa aktualizacja pola w 100 modelach
12	Warunkowa aktualizacja relacji w 100 modelach

4.3. Aplikacja testowa

Framework Django sam wykonuje znaczną część procesu budowy aplikacji. Implementacja aplikacji testowej ograniczyła się do podstawowych kroków:

1) skonfigurowanie bazy danych,
2) zdefiniowanie modeli
3) wykonanie migracji.
4) wdrożenie panelu administratora,
5) implementacja scenariuszy testowych.

Do przeprowadzenia badań zdefiniowano dwa modele. Pierwszy z nich (przykład 1), to model *Team.* Model ten reprezentuje drużynę i zawiera jedno pole *name* typu *Charfield* z argumentem *max_length* o wartości 255, co przekłada się na wartość typu string o maksymalnej długości 255 znaków. Zdefiniowanie metody *__str__* w tym modelu sprawi, że w momencie rzutowania obiektu klasy *Team* na typ string, zostanie zwrócona nazwa drużyny.

Przykład 1. Model drużyny Team

```
class Team(models.Model):
   name = models.CharField(max_length=255)
   def __str__(self):
      return self.name
```

Przykład 2 przedstawia fragment kodu reprezentującego model meczu *Match.* W klasie *Match* zostały zdefiniowane pola *home_team* oraz *away_team*, które są odwołaniami (kluczami obcymi) do modeli drużyn. Liczby zdobytych punktów przez drużyny zostaną zapisane w polach *home_team_score* oraz *away_team_score*. Pola te mają zdefiniowaną regułę pozwalającą na zapisanie dodatnich

liczb całkowitych lub *null* (w przypadku gdy mecz jeszcze się nie odbył). Data meczu zapisywana jest w polu *date*. Pole *finished* określa, czy dany mecz już się odbył - jest to pole typu *boolean*. Pole typu *float* definiuje mnożnik punktowy z domyślną wartością 1.

Dodatkowo zdefiniowano walidację tworzonego meczu, która sprawdza, czy drużyna gospodarzy nie jest jednocześnie drużyną gości. W przypadku próby utworzenia takiego obiektu - aplikacja zwróci błąd. Rzutowanie na typ string obiektu klasy *Match* zwróci nazwę drużyny gości oraz gospodarzy.

Przykład 2. Model meczu Match

```
class Match(models.Model):
    home_team = models.ForeignKey(
        Team,
        on_delete=models.CASCADE,
        related_name='%(class)s_home_team'
    )
    away_team = models.ForeignKey(
        Team,
        on_delete=models.CASCADE,
        related_name='%(class)s_away_team'
    )
    home_team_score = models.PositiveSmallIntegerField(
        blank=True,
        null=True
    )
    away_team_score = models.PositiveSmallIntegerField(
        blank=True,
        null=True
    )
    date = models.DateTimeField(
        null=True
    )
    finished = models.BooleanField(
        default=False
    )
    reward_multiplier = models.FloatField(
        default=1
    )
   def clean(self, *args, **kwargs):
        from django.core.exceptions import ValidationError
        if self.home_team == self.away_team:
            raise ValidationError("Error")
        super(Match, self).clean(*args, **kwargs)
    def save(self, *args, **kwargs):
        self.full_clean()
        super(Match, self).save(*args, **kwargs)
    def __str__(self):
        return "%s vs %s" \
               % (self.home_team.name, self.away_team.name)
```

Scenariusze zostały wykonane wykorzystując wiersz poleceń. Dla wszystkich scenariuszy utworzono jedną klasę *Command*, która dziedziczy po klasie *BaseCommand*. W Django klasy tworzące komendę, dostępną z wiersza poleceń, umieszcza, się w folderze *management/commands*. Nazwa pliku komendy ma istotne znaczenie ponieważ stanowi ostatni człon nazwy komendy. Przykładowo plik komendy o nazwie *tests.py* tworzy komendę „*python manage.py tests*”.

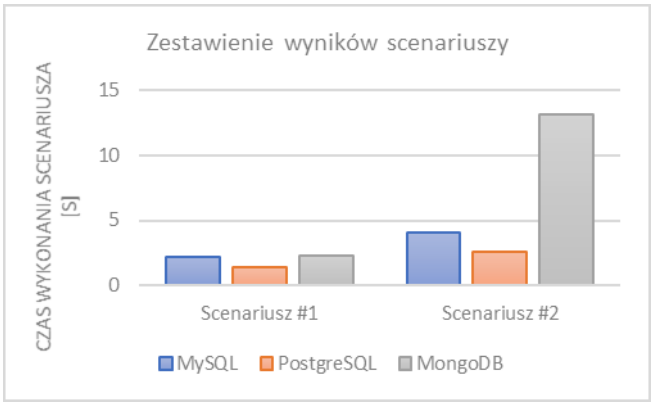
Każdy ze scenariuszy został zdefiniowany w oddzielnej metodzie. Czas poszczególnych scenariuszy mierzony jest za pomocą funkcji *time()* pochodzącej z klasy *time*. Fragment kodu mierzący czas przedstawia przykład 3. Ten sam sposób mierzenia czasu zaimplementowany jest w każdym ze scenariuszy. Implementację jednego ze scenariuszy pokazuje przykład 3.

Przykład 3. Implementacja przykładowego scenariusza

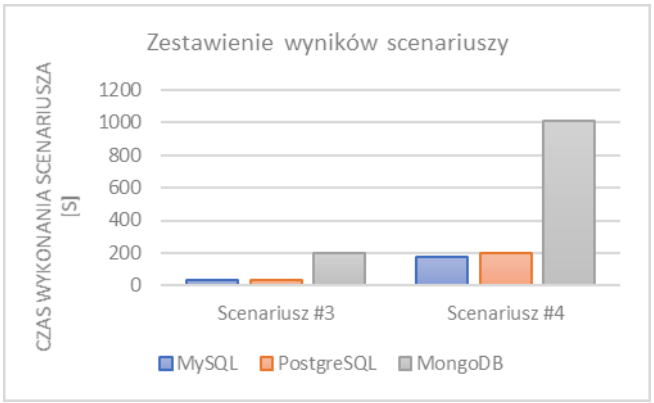
```
def scenario1(self):
scenario = ScenarioInfo(1)
scenario.tries = self.tries
for x in range(0, scenario.tries):
    start_time = time.time()
    team = Team()
    team.name = "New Team"
    team.save(True)
    elapsed_time = time.time() - start_time
    scenario.total_time += elapsed_time
return scenario
```

4.4. Porównanie wydajności

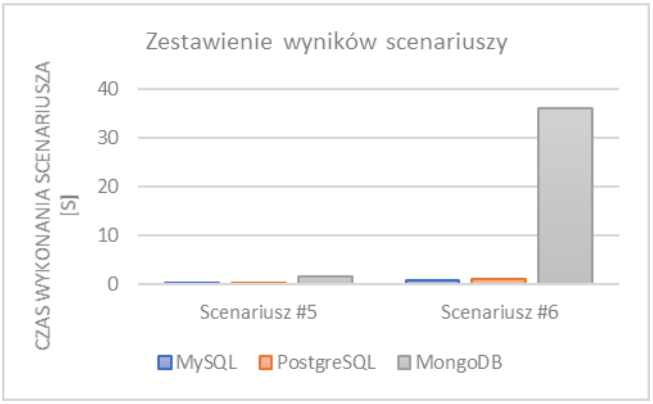
Na rysunkach 1-6 przedstawiono wyniki poszczególnych scenariuszy.



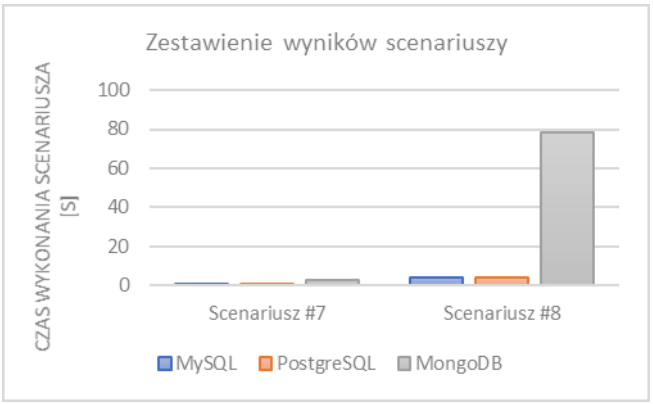
Rys. . Utworzenie jednego modelu bez i z relacjami (scenariusz 1 i 2)



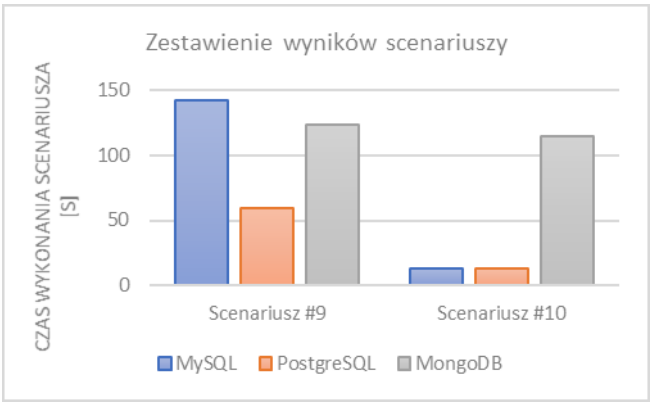
Rys. 2. Utworzenie 100 modeli bez i z relacjami (scenariusz 3 i 4)



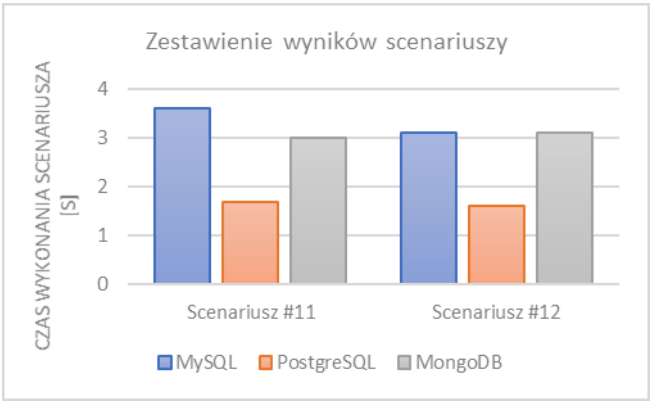
Rys. 3. Pobranie jednego modelu bez relacji i z relacją (scenariusz 5 i 6)



Rys. 4. Pobranie 100 modeli bez relacji i z relacją (scenariusz 7 i 8)



Rys. 5. Warunkowe pobranie modeli bez i z relacją (scenariusz 9 i 10)



Rys. 6. Warunkowa aktualizacja w 100 modelach (scenariusz 11 i 12)

5. Wnioski

Na podstawie wyników badań, można wywnioskować, że MongoDB całkowicie nie nadaje się do aplikacji napisanych w Django. Jedną z przyczyn może być fakt, że baza MongoDB nie jest oficjalnie wspierana przez twórców Django. Implementacja MongoDB w zaprezentowanej aplikacji testowej została zrealizowana dzięki zewnętrznej paczce *djongo*. Działanie tej paczki polega na tłumaczeniu zapytań SQL, wygenerowanych przez ORM na składnię MongoDB, co wiąże się z utratą wydajności przy każdym zapytaniu. Kolejnym istotnym czynnikiem jest to, że MongoDB nie wspiera kluczy obcych, co w efekcie . w porównaniu do relacyjnych systemów (na przykładzie scenariuszy, w których operacje związane były z relacjami) znacznie wydłuża czas dostępu do danych.

Przewaga PostgreSQL w większości scenariuszy nie oznacza iż jest to jedyny słuszny system w przypadku aplikacji internetowych. W przypadku aplikacji testowej wykonanej na potrzeby tego artykułu, różnica w szybkości między PostgreSQL, a MySQL wynosi około 17%. Jednak gdyby w aplikacji położyć nacisk na tworzenie modeli z relacjami - przewaga PostgreSQL mogłaby znacząco zmaleć. Największą przewagę PostgreSQL (ponad dwukrotnie mniejszym od MySQL) uzyskał w scenariuszu, który polegał na warunkowym pobieraniu modeli z czasem.

Podsumowując - osoba poszukująca najbardziej wydajnego systemu bazodanowego do pracy z aplikacją Django powinna wziąć pod uwagę to, czy aplikacja będzie częściej wykonywała operacje odczytu, czy zapisu. W przypadku przewagi operacji odczytu, a zwłaszcza odczytu ograniczonego warunkami, lepszym rozwiązaniem może okazać się PostgreSQL, natomiast przewaga operacji zapisu do bazy danych stawia MySQL na lepszej pozycji.

Wyniki uzyskane z testów pozwoliły potwierdzić, postawioną na wstępie tezę.

Literatura

[1] https://www.devsaran.com/blog/history-web-application-development [29.11.2018]

[2] http://mst.mimuw.edu.pl/lecture.php?lecture=bad&part=Ch1 [29.11.2018]

[3] https://www.thoughtfulcode.com/orm-active-record-vs-data-mapper/ [29.11.2018]

[4] https://db-engines.com/en/ranking [29.11.2018]

[5] https://bytescout.com/blog/2014/09/mongodb-history-and-advantages.html [29.11.2018]

[6] Chodorow K.: MongoDB: The Definitive Guide, Second Edition, O'Reilly Media, 2013.

[7] https://www.oracle.com/MySQL/ [29.11.2018]

[8] https://bytescout.com/blog/2014/09/mongodb-history-and-advantages.html [29.11.2018]

[9] Obe R., Hsu L.: PostgreSQL: Up and Running, Second Edition, O'Reilly Media, 2015.