Autorzy: Michał Mrowczyk, Damian Kudas

Technologia: Hibernate + PostgreSQL

# Ładowanie danych do bazy

***UWAGA****:* Kolejność ładowania danych z xml’i ma znaczenie. Nie możemy np. ładować zamówień bez uprzedniego załadowania pracowników, gdyż dla każdego zamówienia musimy mieć referencję na wiersz w tabeli pracowników, reprezentujący pracownika. Dlatego też zawsze przed ładowaniem zamówień, ładujemy uprzednio z pamięci do bazy dane zapisane w xml’ach: customers.xml, employees.xml, categories.xml, suppliers.xml, shippers.xml, products.xml

Zmierzyliśmy wydajność ładowania danych do bazy w zamówieniach na sekundę, licząc tą wydajność co 1000 zamówień. Wykres przedstawia wyniki:

Jak widać na wykresie ciężko dopatrzyć się jakichś większych regularności. Jedyne co obserwowaliśmy dla kilku podjętych prób to dosyć powolny start, następnie przejście do maksimum, a pod koniec lekki spadek wydajności i stabilizacja.

Średni czas trwania całej operacji wynosił ok. **213.013 s**, czyli ponad 3.5 minuty.

Ciekawostką jest fakt, że z wykresu wynika średnia wydajność ok. 400 zamówień/sekundę. Biorąc to dosłownie całość powinna zakończyć się, po około 1.5 minuty… Prawda jest jednak taka, że większość czasu została spędzona w funkcji:

session.getTransaction().commit();

To tutaj Hibernate wykonuje całą pracę związaną z zapisywaniem danych do bazy.

Powyższe rezultaty osiągnęliśmy kaskadowo zapisując elementy zamówienia – OrderDetails:

@OneToMany(mappedBy="Order", cascade=CascadeType.*ALL*)

**private** Set<OrderDetails> OrderDetailss = **new** HashSet<OrderDetails>();

Jako, że zarówno zamówienia jak i detale zamówień są przechowywane w pamięci naszego programu, spróbowaliśmy innego podejścia:

„Zapisz wszystkie zamówienia bez kaskadowego zapisywania detali zamówień, następnie zapisz wszystkie detale zamówień”

Wynikowy czas zapisu to:

**132.972 s**

Usunięcie kaskadowego zapisywania, spowodowało blisko dwukrotne przyspieszenie działania aplikacji. Teraz nie ma zauważalnego narzutu na **session.save(obj).** Większość czasu nadal jest spędzana w metodzie: **commit()**

Na potwierdzenie naszych słów wykres:

Oczywiście małe oszustwo jest takie, że wykres ten nie pokazuje czasu zapisywania elementów zamówienia – OrderDetails. Jest on jednak porównywalny z zapisywaniem samych zamówień, a zatem bardzo mały. (Mówiąc o zapisywaniu mamy na myśli użycie metody **session.save(obj)** co jednak w rzeczy samej do bazy niczego nie zapisuje.

**REASUMUJĄC:**

Niewłaściwe korzystanie z kaskadowego zapisywania w Hibernate może doprowadzić do znacznego spadku wydajności. Dlatego jeśli mamy możliwość (np. posiadamy całe dane zapisane jako kolekcje w pamięci), to zapisujmy obiekty osobno.

# Przygotowanie zestawień dotyczących danych

## Ile zamówień z każdego z krajów zostało zrealizowanych

Zapytanie:

select c.country, count(o.orderID) from orders o join customers c on

c.customerID = o.customerID group by c.country

Średni czas wykonania zapytania to:

**253 ms**

## Jaki był średni czas realizacji zamówienia w każdym roku

Zapytanie:

select avg(ShippedDate - OrderDate), DATE\_PART('year', ShippedDate) from orders group by DATE\_PART('year', ShippedDate) order by DATE\_PART('year', ShippedDate)

Średni czas wykonania zapytania to:

**290 ms**

## Ile sztuk produktów od każdego z dostawców udało się sprzedać

Zapytanie:

select s.CompanyName, sum(od.Quantity) from suppliers s inner join products p on s.SupplierID = p.SupplierID inner join orderdetails od on p.ProductID = od.ProductID group by s.CompanyName

Średni czas wykonania zapytania to:

**405 ms**

## Jaka kwota zamówień była zgłaszana w każdy z dni tygodnia

Zapytanie:

select extract(dow from o.OrderDate), sum(od.UnitPrice \* od.Quantity) from orderdetails od join orders o on o.OrderID = od.OrderID group by extract(dow from o.OrderDate) order by extract(dow from o.OrderDate)

Średni czas wykonania zapytania to:

**575 ms**

## Jaka była wartość produktów zamówionych z każdego z krajów w każdym roku; chodzi o kraj zamawiającego

Zapytanie:

select c.Country, date\_part('year', o.OrderDate), sum(od.Quantity \* Od.UnitPrice) from

customers c join orders o on c.CustomerID = o.CustomerID join orderdetails od on o.OrderID = od.OrderId group by 1,2

Średni czas wykonania zapytania to:

**4220 ms**

## Jaka była średnia wartość sztuki produktu dla każdego ze spedytorów w każdym roku

Zapytanie:

select s.CompanyName, date\_part('year', o.ShippedDate), sum(od.UnitPrice \* od.Quantity) / sum(od.Quantity)

from shippers s join orders o on s.ShipperID = o.ShipVia join orderdetails od on o.OrderID = od.OrderID

group by 1,2

Średni czas wykonania zapytania:

**8322 ms**

**Uwaga:**

Porównując czasy wykonywania zapytań select o dane, dochodzimy do wniosku, że zapytania wykonywane z poziomu Eclipse (Java) są równie wydajne jak te wykonywane bezpośrednio z poziomu pgAdmin (narzędzie graficzne dla administratorów baz PostgreSQL). Jeśli więc szukać poprawy czasu wykonywania takich zapytań, to znajdziemy ją na poziomie schematu bazy danych (np. tworzenie nowych indeksów dla często poszukiwanych kolumn), czy też konfiguracja samej bazy Postgres.