STUDIA INFORMATICA 2009
Volume 30 Number 3A (85)

Krzysztof DOBOSZ, Roman SIERADZKI
Politechnika Śląska, Instytut Informatyki

PORÓWNANIE EFEKTYWNOŚCI DOSTĘPU DO BAZ DANYCH W WYBRANYCH MECHANIZMACH OPARTYCH NA MASZYNIE WIRTUALNEJ JAVY

Streszczenie. Artykuł prezentuje wyniki badań nad efektywnością dostępu do baz danych z wykorzystaniem wybranych mechanizmów bazujących na maszynie wirtualnej Javy. Porównano ze sobą: bezpośredni dostęp przez sterownik JDBC, automatyczne mapowanie obiektowo-relacyjne oraz wykorzystanie obiektów Enterprise Java Bean. Wyniki najbardziej interesujących porównań przedstawiono na ilustracjach.

Słowa kluczowe: utrwalanie danych, JDBC, Hibernate, EJB

COMPARISON OF EFFECTIVENESS OF DATABASE ACCESS WITH CHOSEN MECHANISMS USING JAVA VIRTUAL MACHINE

Summary. The paper presents results of researches on the field of database access with chosen mechanisms using Java Virtual Machine. Three deferent methods are compared with themselves: direct access by the JDBC driver, automatic object-relational mapping, and using Enterprise Java Bean objects. Results of the most interesting comparisons are shown on illustrations.

Keywords: data persistence, JDBC, Hibernate, EJB

1. Przegląd metod dostępu do baz danych

Język Java jest obecnie jednym z najprężniej rozwijających się języków programowania. Wraz z nim pojawiły się także implementacje różnych mechanizmów, które wspomagają wiele aspektów programowania [1]. Jednym z zagadnień, którym interesują się programiści, jest realizacja komunikacji z bazą danych. Można ją wykonać na wiele sposobów. Ponieważ nie sposób było porównać wszystkich możliwych rozwiązań, do badań wybrano więc trzy

implementacje reprezentujące najbardziej różniące się między sobą mechanizmy: JDBC, EJB i Hibernate. Szczegółowy opis badań został zawarty w pracy dyplomowej magisterskiej [2], zaś niniejszy artykuł prezentuje tylko najbardziej istotne wyniki.

1.1. Interfejs JDBC

JDBC (ang. *Java DataBase Conectivity*) [3] – jest najprostszym, a jednocześnie najbardziej znanym interfejsem programistycznym pozwalającym na bezpośrednią komunikację z bazą danych. Producent serwera baz danych dostarcza zwykle sterownik zaimplementowany zgodnie ze zorientowaną obiektowo specyfikacją JDBC. Umożliwia on w prosty sposób konstruowanie zapytań w języku SQL i umieszczanie ich na poziomie języka Java. Pozwala on więc na łatwe połączenie płaszczyzny działania algorytmów i dostępu do dużych struktur danych. Korzystając z JDBC, nie ma potrzeby instalowania żadnych dodatkowych narzędzi, z wyjątkiem sytuacji, w której projektant oprogramowania zdecyduje się na trójwarstwowy model komunikacji z bazą danych [4].

1.2. Specyfikacja EJB

EJB (ang. *Enterprise JavaBean*) [5] jest podstawową specyfikacją platformy *Java EE* [6] związaną z dostępem do baz danych. Umożliwia ona tworzenie komponentów programowych tzw. ziaren, które mogą być instalowane i uruchamiane na serwerze aplikacyjnym i udostępniane zdalnie przez protokół RMI-IIOP. Z punktu widzenia realizacji dostępu do bazy danych rozróżniamy dwa typy ziaren:

- CMP (ang. *Container Managed Persistence*) – trwałością i związanymi z tym aspektami (m.in. obsługa błędów, realizacja transakcji) zarządza kontener ziaren dostarczony przez producenta serwera aplikacji,
- BMP (ang. *Bean Managed Persistence*) – wszystkimi zagadnieniami zawiązanymi z zarządzaniem trwałością musi zająć się programista.

Serwer aplikacji wymaga zainstalowania sterownika JDBC. Badana specyfikacja w wersji 2.1 jest zdecydowanie najbardziej skomplikowana spośród analizowanych narzędzi i zmusza programistę do największego wkładu czasowego w opracowywanie aplikacji.

1.3. Biblioteka *Hibernate*

Hibernate [7] jest zaawansowaną biblioteką wspomagającą programowanie aplikacji bazodanowych. Zapewnia ona mapowanie pomiędzy relacyjną bazą danych a obiektami w ujęciu programistycznym. Do opisu danych wykorzystuje ona standard XML, dzięki czemu pozwala na rzutowanie obiektów w dowolnym języku programowania, na wiersze w bazie da-

nych, co czyni ją rozwiązaniem uniwersalnym i przenośnym. Umożliwia ona wykorzystanie zapytań zdefiniowanych zarówno w języku SQL, jak i obiektowo zorientowanym HQL. Na niskim poziomie wykorzystuje ona sterownik JDBC. Możliwe jest umieszczenie części odpowiedzialnej za odwzorowania obiektowo-relacyjne zarówno na serwerze aplikacji, jak i w części klienckiej. Udostępnia ona bardzo szeroką gamę ustawień i właściwości pozwalających na wydajną konfigurację i efektywne działanie. Jest to biblioteka otwarta (ang. *open source*).

2. Wykorzystane narzędzia

Do celu przeprowadzenia badań wykorzystano serwer aplikacyjny *JBoss* oraz serwer bazy danych *Microsoft SQL Server 2000* z przykładową bazą danych *Northwind Traders*.

2.1. Serwer aplikacji

JBoss jest serwerem aplikacji z otwartym dostępem do swego kodu źródłowego, pracującym na platformie *Java EE*. Implementuje on pełną paletę specyfikacji, takich jak: EJB, JPA, JMS, JTS/JTA, JSP, JNDI, JMX, WebServices, a także zaawansowane mechanizmy podziału na klastry. Aby wdrożyć aplikację w serwerze JBOSS, wystarczy przekopiować do odpowiedniego katalogu plik w formacie właściwym dla danej technologii, jak na przykład *jar, war*, lub *ear*.

Serce serwera oparte jest na mechanizmie JMX. Oznacza to, że wdrażany komponent automatycznie jest zaliczany do elementów, które mogą być przedmiotem standardowych operacji zarządzania. Każda usługa wdrożona na serwerze *JBoss* nazywana jest komponentem zarządzanym. Serwer udostępnia możliwość zarządzania usługami za pośrednictwem sieci przez połączenie z odpowiednim adresem URL. Każdy protokół obsługiwany jest przez oddzielny moduł, do którego trafiają żądania. Dzięki temu każdy nowo wdrożony moduł zostaje automatycznie powiązany ze wszystkimi dostępnymi usługami.

2.2. Baza danych

W przeprowadzonych testach wykorzystano serwer bazodanowy *Microsoft SQL Server 2000* oraz przykładową bazę danych *Northwind Traders* dostarczaną wraz z tym narzędziem. *SQL Server 2000* jest kompletnym rozwiązaniem bazodanowym i analitycznym do szybkiego tworzenia aplikacji przeznaczonych dla Internetu. Oferuje on ponadto możliwość podziału obciążenia, co zwiększa jego wydajność. Zaawansowane funkcje zarządzania pozwalają automatyzować rutynowe zadania.

Baza *Northwind Traders* jest demonstracyjną bazą danych dostarczaną wraz z *MS SQL Server 2000*. Zawiera ona informacje o sprzedaży prowadzonej przez fikcyjną firmę o nazwie „Northwind Traders", zajmującej się importem i eksportem produktów spożywczych z całego świata. W bazie rejestrowane są zamówienia wykorzystywane do wystawiania faktur oraz do prowadzenia analityki sprzedaży. Jest to baza o niezbyt dużym stopniu skomplikowania, która pozwala na wywoływanie operacji bazodanowych o różnorakim poziomie zagnieżdżeń. Zaletą bazy jest to, że jest już zaprojektowana oraz wypełniona przykładowymi danymi, dzięki temu doskonale nadaje się do przeprowadzania na niej powtarzalnych testów.

3. Testy

Testy przeprowadzone zostały na komputerze z procesorem Intel Centrino 1,73 GHz wyposażonym w 1,5 GB RAM. Podczas badań działania *Hibernate* i EJB zarówno oprogramowanie klienta, jak i serwera znajdowały się na tej samej maszynie, aby uniknąć przekłamań związanych z przepustowością sieci.

W celu zbadania efektywności dostępu do bazy danych należało wybrać kryteria porównawcze oraz przygotować w języku SQL odpowiednie zapytania.

3.1. Kryteria testów

W czasie przeprowadzania testów brano po uwagę różne kryteria, w których otrzymane wyniki można klasyfikować oraz porównywać. Podczas analizy wyników badań kierowano się następującymi kryteriami:

- zapytania o różnym stopniu skomplikowania i różnej liczbie powiązanych relacjami tabel,
- transakcje opakowujące zapytania, z wykorzystaniem różnych mechanizmów udostępnianych przez badane technologie (należy tutaj zaznaczyć, że zapytania z wykorzystaniem polecenia SELECT nie są najlepszym przykładem do konstruowania transakcji, ale dzięki nim możliwe jest wykonywanie bardzo dużej liczby operacji na bazie danych bez konieczności kontroli jej rozmiaru oraz poprawności danych),
- połączenia z bazą danych wykonywane w dwóch trybach: wszystkie w jednym połączeniu bądź też nawiązywanie nowego połączenia dla każdej operacji, a po jej wykonaniu jego zwalnianie,
- badanie czasu wykonania różnej liczby operacji.

3.2. Zapytania testowe

Testy przeprowadzono dla czterech zapytań, każde z nich charakteryzuje się inną złożonością:

- najprostsze, ale jednocześnie zwraca najwięcej danych,

```
select * from Customers
```

- zwraca tylko jeden wiersz, ograniczone jest tylko jednym warunkiem,

```
select * from Customers where CompanyName = 'Alfreds Futterkiste'
```

- zapytanie o średnim poziomie skomplikowania zwracające dużą liczbę wyników,

```
select distinct c.CompanyName from Customers c ,[Order Details] od, Orders o
where c.CustomerID=o.CustomerID and od.OrderID=o.OrderID and od.Quantity>12
```

- bardzo skomplikowane zapytanie zwracające dużo wierszy wynikowych.

```
select CU.CompanyName,CU.Country,sum(OD.Quantity) as Quantity from Customers CU,
Orders O, [Order Details] OD,Products P where CU.CustomerID = O.CustomerID and
O.OrderID = OD.OrderID and OD.ProductID = P.ProductID and P.UnitsInStock > 10
group by CU.CompanyName,CU.Country
```

Zapytania powtarzane były w seriach po 10, 100, 1000 oraz 10000 w jednym zestawie, testowane zarówno dla jednego połączenia, jak i dla osobnych połączeń. Każda technologia przetestowana została pod względem wydajności zwykłych zapytań, czyli takich, które nie są wykonywane ani w transakcji, ani zapytaniach wykorzystujących różne tryby działania transakcji.

4. Wyniki

Podczas przeprowadzania testów starano się zapewnić jak najbardziej jednolite środowisko testowe dla wszystkich badanych technologii. Pomiary dla każdego z testów przeprowadzano pięciokrotnie, po czym dwa skrajne wyniki odrzucano, a z pozostałych obliczano średnią arytmetyczną. Zestawienia wyników zamieszczone w niniejszym artykule dotyczą testów, których wykonywano 10 000 powtórzeń każdego zapytania bez udziału mechanizmu transakcji oraz z jego wykorzystaniem.

4.1. Testy bez udziału mechanizmu transakcji

W ramach testów bezpośredniego dostępu do bazy danych poprzez sterownik JDBC wykonano szereg zapytań o różnych poziomach skomplikowania (podano je w rozdz. 3.2) z zastosowaniem następujących mechanizmów:

- zwykłe zapytanie SQL,
- zapytanie preinterpretowane,
- procedura składowana.

Podczas testów okazało się, że wykorzystywany sterownik JDBC dla serwera bazy danych *SQL Server 2000* firmy *Microsoft* działał niepoprawnie podczas wywoływania procedur składowanych, co powodowało, że uzyskane czasy wykonania były nieporównywalne z innymi. Dlatego też wyników tych nie umieszczono w zestawieniu.

Testy biblioteki *Hibernate* przeprowadzono dla zapytań zdefiniowanych w dwóch językach zapytań:

- obiektowym HQL,
- SQL, którego wyniki rzutowano potem na odpowiednie obiekty biblioteki.

Wszystkie testy wykonane zostały ponadto dla dwóch lokalizacji biblioteki mapowania obiektowo-relacyjnego:

- po stronie aplikacji klienta,
- po stronie serwera aplikacji.

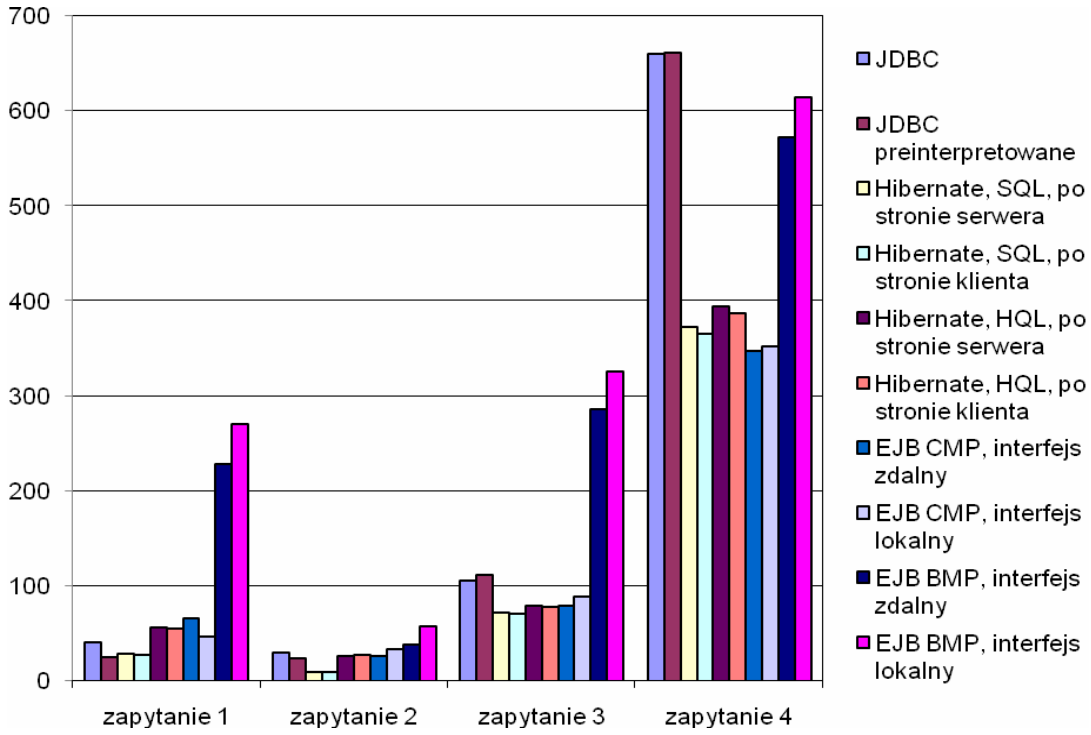
Specyfikacja EJB umożliwia wykonywanie najbardziej rozbudowanych testów. Po pierwsze, umożliwia dostęp do ziaren za pomocą jednego z dwóch interfejsów: zdalnego lub lokalnego. Po drugie, pozwala na dwa sposoby zarządzania trwałością w ziarnach: CMP i BMP. Testy wykonano więc dla czterech konfiguracji.

Wszystkie testy wykonano zarówno dla pętli zapytań wykonywanych w ramach jednego połączenia (rys. 1), jak i dla każdego zapytania wykonanego w osobnym połączeniu (lub osobnym kontekście sesji w przypadku *Hibernate* i *EJB*) (rys. 2). Liczba wykonanych zapytań w każdym ze zilustrowanych zestawień wyników wynosiła 10 000. Jednostką czasu na osi pionowej wszystkich wykresów są sekundy.

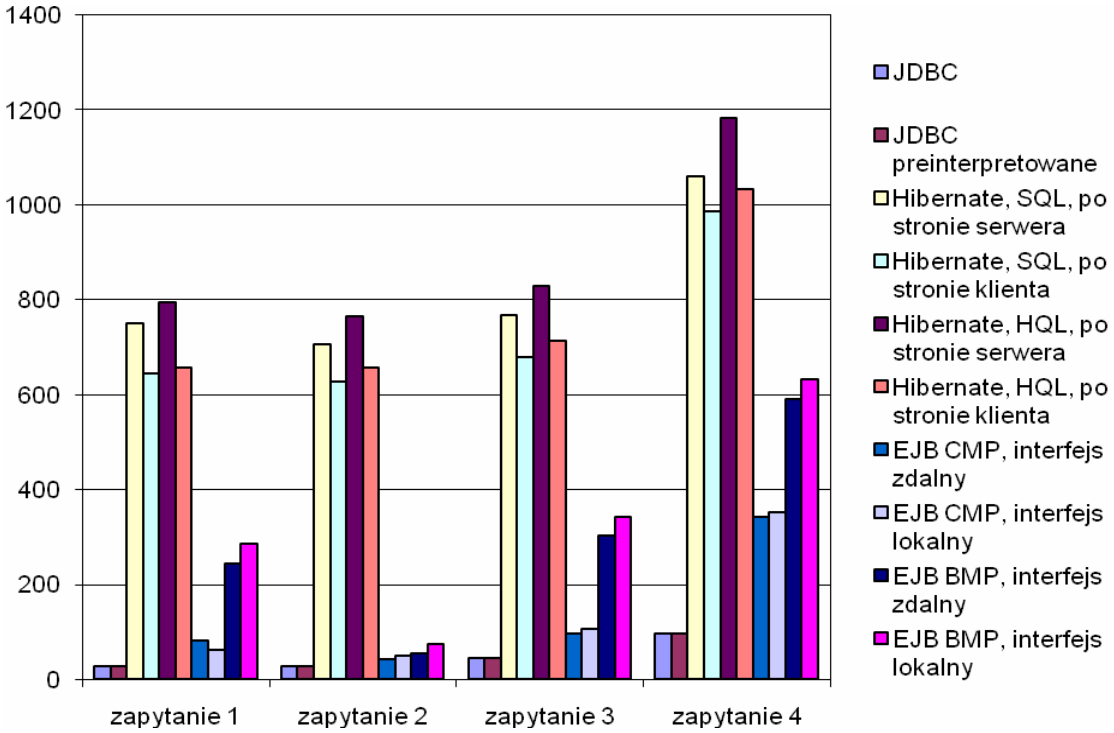
Na podstawie wyników zebranych na rys. 1 trudno jednoznacznie stwierdzić, która technologia dla zapytań wykonywanych w ramach jednego połączenia (sesji, kontekstu) daje najlepsze rezultaty. Dla zapytań nr 1, 2 i 3 najlepiej wypadła biblioteka *Hibernate* wykorzystująca język SQL i to bez względu na fakt, czy kod komunikujący się z serwerem bazy danych był częścią aplikacji klienckiej, czy serwera aplikacji. W przypadku najprostszego zapytania nr 1 konkurować z nią mogły tylko zapytania preinterpretowane w JDBC.

W przypadku najbardziej złożonego zapytania nr 4 lepsze wyniki niż za pomocą wspomnianej biblioteki osiągnięto z wykorzystaniem ziaren EJB CMP. We wszystkich testach pętli zapytań wykonywanych podczas jednokrotnego połączenia najgorsze wyniki otrzymano dla technologii EJB, w której ziarna BMP wykorzystywały interfejs lokalny (w przypadku gdyby oprogramowanie klienta znajdowało się na innej maszynie niż serwer, prawdopodobnie gorszy wynik uzyskano by dla ziaren wykorzystujących interfejs zdalny). Gorsze rezulta-

ty otrzymano jedynie w przypadku zapytania najbardziej złożonego podczas bezpośredniego dostępu przez JDBC.



Rys. 1. Wyniki testów: pojedyncze połączenie, bez transakcji
Fig. 1. Tests results: single connection, no transactions



Rys. 2. Wyniki testów: oddzielne połączenia, bez transakcji
Fig. 2. Tests results: separated connections, no transactions

Wyniki operacji wykonywanych w pętli połączeń zobrazowano na rys. 2. Wyniki dla bezpośredniego dostępu za pomocą interfejsu JDBC uzyskane dla większej liczby połączeń nie mogą być brane pod uwagę. Podczas testów nastąpił wysoki odsetek odrzuconych połączeń – w sumie przy 10000 próbach odrzucono 8017, więc wyniki zaprezentowane na wykresie są zdecydowanie zaniżone.

Analizując wykres z rys. 2, bez trudu można dostrzec, że najgorsze wyniki uzyskano za pomocą zapytań HQL wykonywanych poprzez *Hibernate* za pośrednictwem serwera *JBoss*. Technologia *Hibernate* korzysta z bardzo dużej liczby bibliotek i klas pomocniczych, które są ładowane podczas tworzenia sesji, co prawdopodobnie powoduje narzuty czasowe, które w znaczący sposób wpływają na pogorszenie wydajności takiej operacji. Innym powodem uzyskiwania słabszych czasów dla *Hibernate* jest spory narzut czasowy wprowadzany przez serwer *JBoss* podczas połączenia do bazy danych.

Natomiast najlepsze czasy w większości przypadków (chociaż nie jest to regułą) uzyskano ponownie dla ziaren EJB CMP. Nasuwa się więc wniosek, że wykorzystanie kontenera do zarządzania zapytaniami daje najlepsze wyniki, jeżeli chodzi o dużą liczbę połączeń. Wykorzystanie serwera *JBoss* wraz z technologią *Hibernate* daje najsłabsze wyniki, a z EJB - najlepsze. Rozwiązaniem pośrednim byłoby tutaj wykorzystanie *Hibernate* do mapowania relacyjnej bazy danych na obiekty EJB CMP. Takie rozwiązanie jest także dosyć powszechnie stosowane, jednak nie zostało uwzględnione w ramach niniejszej pracy.

4.2. Testy z wykorzystaniem mechanizmu transakcji

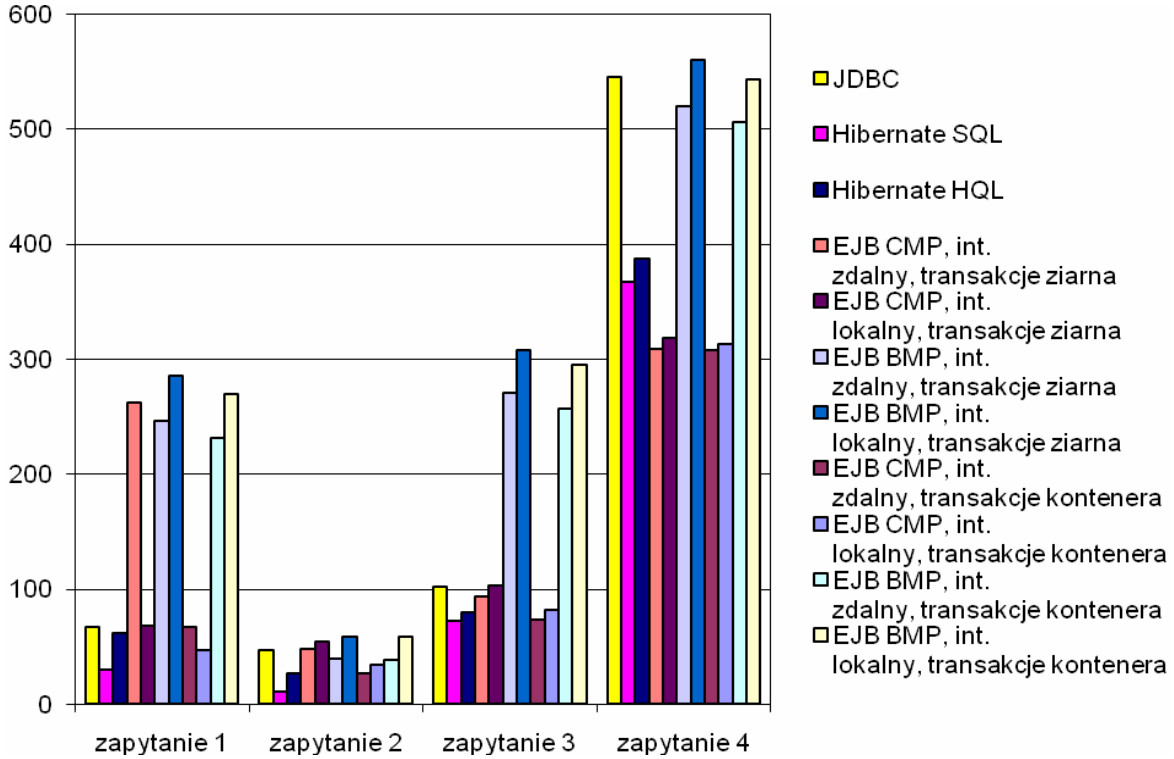
Transakcje są jedną z podstawowych funkcjonalności wykorzystywanych przez programistów podczas tworzenia aplikacji łączących się z bazą danych. Ideą transakcji jest wykonanie kilku operacji na bazie danych w ramach jednej operacji użytkownika, a w przypadku niepowodzenia jednej z nich - odwołanie wszystkich. W praktyce zapytań typu SELECT nie opakowuje się w transakcję, lecz dla celów testowych takie rozwiązanie jest wystarczające.

Transakcje mogą być wykonywane z wykorzystaniem czterech poziomów izolacji [4]. Do badań wybrano dwa skrajne poziomy: READ_UNCOMMITED zapewniający minimalny poziom bezpieczeństwa i jednocześnie najszybsze działanie oraz SERIALIZABLE dający największy narzut czasowy, ale za to zapewniający najwyższy poziom bezpieczeństwa i synchronizacji danych. Każde zapytanie wykonywane zostało w ramach pojedynczego połączeniu oraz w ramach pętli osobnych połączeń, w których wykonywano pojedyncze zapytanie.

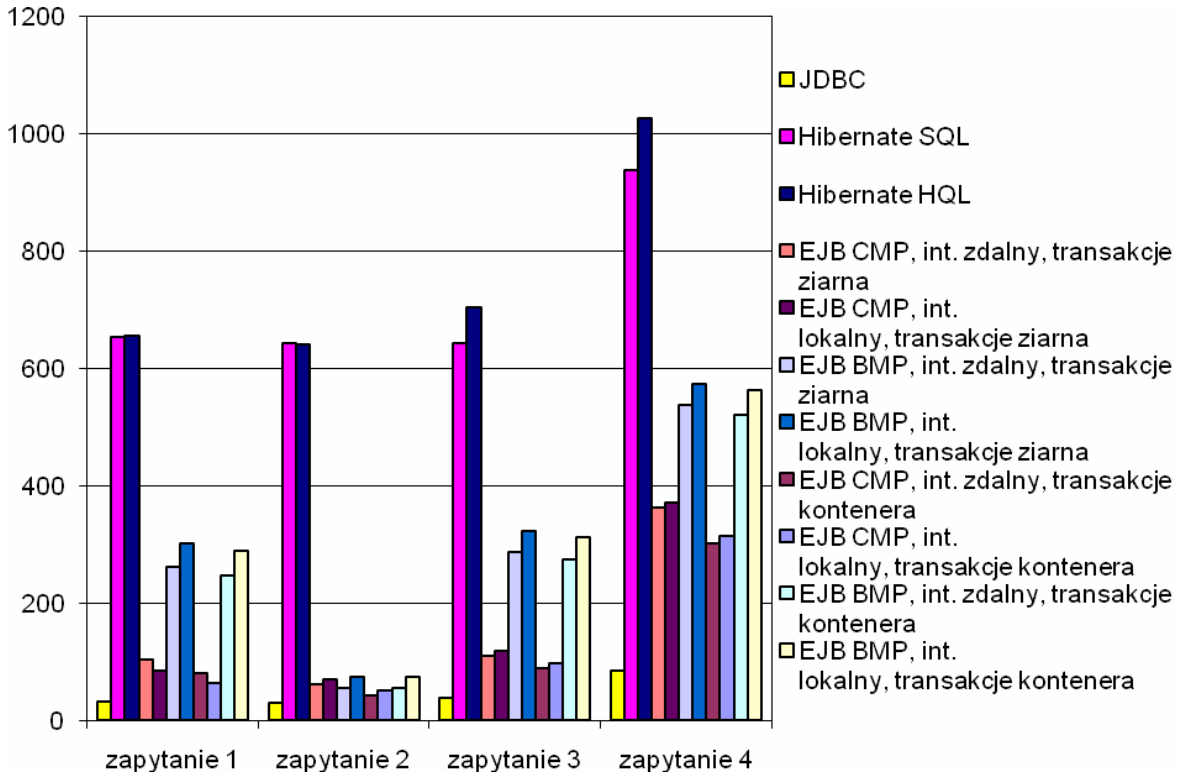
Zgodnie z przewidywaniami, zapytania opakowane w transakcje z poziomem izolacji READ_UNCOMMITED mają najkrótsze czasy wykonania, a zapytania opakowane w transakcje z poziomem izolacji SERIALIZABLE - najdłuższe. Można jednak zauważyć, że zależnie od badanego mechanizmu poziom izolacji ma mniejszy lub większy wpływ na czas wy-

konania operacji. Można stwierdzić, że różnice w wykonaniu transakcji o różnym poziomie izolacji zależne są od sposobu optymalizacji ich wykonania w narzędziach opartych na badanych mechanizmach. Najmniejsze różnice pod tym względem zanotowano w przypadku technologii *Hibernate*, natomiast największe rozbieżności - dla JDBC. Realizacja transakcji z wykorzystaniem JDBC nie jest praktycznie wcale optymalizowana, co wpływa na słabe wyniki tej technologii. Odpowiednie wykresy pokazują wyniki transakcji wykonanych z poziomem izolacji READ_UNCOMMITED (rys. 3) i SERIALIZABLE (rys. 4).

Jak widać na zamieszczonych w niniejszym artykule wykresach, zarówno dla poziomu READ_UNCOMMITED, jak i SERIALIZABLE najszybciej wykonywana była operacja realizowana z wykorzystaniem biblioteki *Hibernate*. Zarówno w przypadku HQL, jak i natywnego SQL (te wykonywało się trochę wolniej) czasy były znacząco lepsze od pozostałych technologii. Dosyć dobrze wypadły także wyniki technologii EJB, a w szczególności ziaren CMP i transakcji realizowanych przez kontener. Najwolniej zdecydowanie wypadła technologia EJB, w której wykorzystano transakcje realizowane bezpośrednio w kodzie ziarna, czyli ziarno BMP z transakcją realizowaną przez JDBC. Transakcje wykonywane w ramach ziarna BMP uzyskały w miarę akceptowalny wyniki jedynie dla zapytania nr 2, które było stosunkowo proste i zwracało najmniej danych.



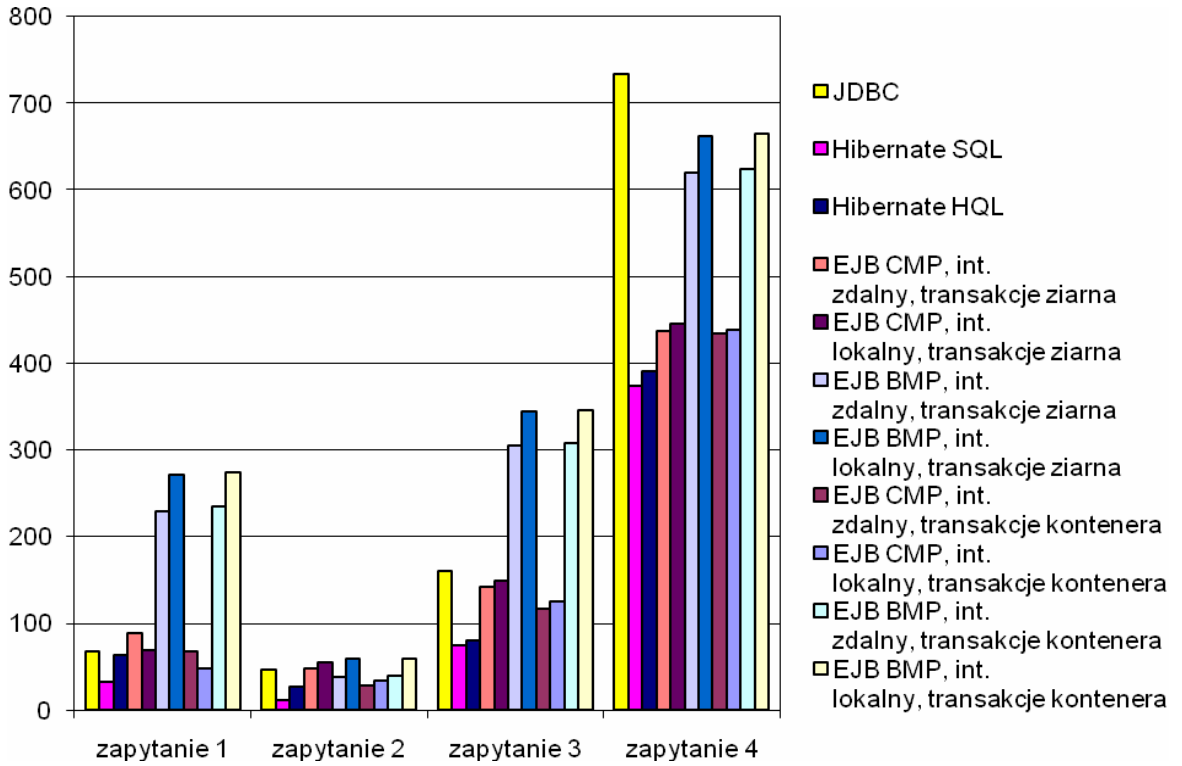
Rys. 3. Wyniki testów: pojedyncze połączenie, poziom izolacji READ_UNCOMMITED
Fig. 3. Tests results: single connection, READ_UNCOMMITED isolation level



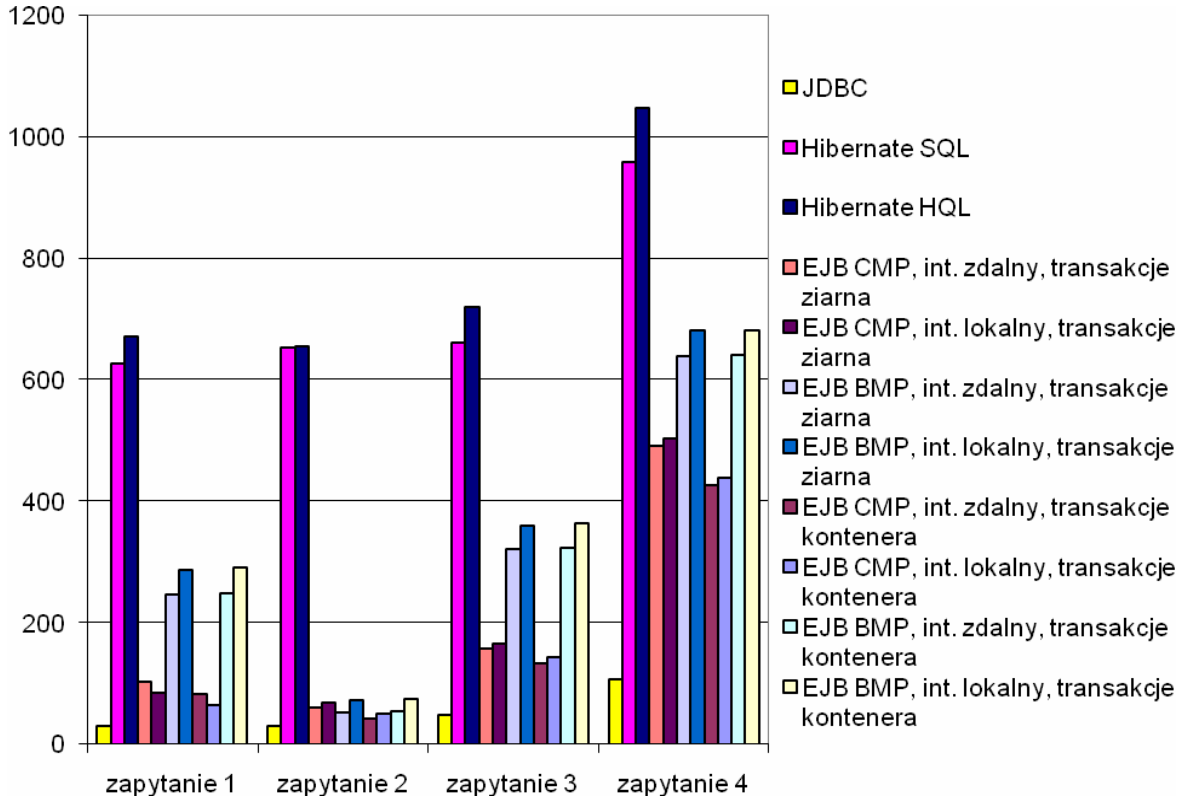
Rys. 4. Wyniki testów: pojedyncze połączenie, poziom izolacji SERIALIZABLE
Fig. 4. Tests results: single connection, SERIALIZABLE isolation level

W przypadku transakcji wykonywanych w pętli oddzielnych połączeń wyniki osiągnięte w ramach przeprowadzonych testów są zupełnie różne od tych uzyskanych w ramach transakcji wykonywanych w jednym połączeniu. W przypadku JDBC znowu wyniki są niewiarygodne, ponieważ serwer odrzucił większość połączeń i dlatego nie należy brać ich pod uwagę. Zgodnie z otrzymanymi rezultatami zdecydowanie najlepsze wyniki osiągnęły ziarna CMP, w których obsługą transakcji zajmował się kontener EJB. Na podstawie wyników nie widać jednak różnic pomiędzy ziarnami implementującymi interfejs lokalny a ziarnami implementującymi interfejs zdalny. Wynika to stąd, że zarówno klient, jak i serwer znajdowali się na tej samej maszynie, co skutkowało tym, że nie wystąpił narzut czasowy związany z przesyłaniem całej operacji na serwer, jak to jest w przypadku ziaren z interfejsem zdalnym.

W przypadku interfejsu lokalnego serwer wykonuje całą operację związaną z bazą danych lokalnie i wysyła do klienta poprzez ziarno implementujące interfejs zdalny tylko wyniki uzyskane w ramach tej operacji. Można więc śmiało stwierdzić, że transakcje realizowane przez ziarna implementujące interfejs lokalny zarządzane przez serwer są jednoznacznie najszybszym rozwiązaniem spośród testowanych w niniejszej pracy technologii. Zamieszczone wykresy (rys. 5 i rys. 6) pozwalają już na pierwszy rzut oka dostrzec opisywane różnice.



Rys. 5. Wyniki testów: 10000 połączeń, poziom izolacji READ_UNCOMMITED
Fig. 5. Tests results: 10000 connections, READ_UNCOMMITED isolation level



Rys. 6. Wyniki testów: 10000 połączeń, poziom izolacji SERIALIZABLE
Fig. 6. Tests results: 10000 connections, SERIALIZABLE isolation level

W tej kategorii, oddzielnych połączeń z udziałem transakcji, najgorzej wypadła technologia *Hibernate*. Jak widać na wykresach, wyniki tej technologii odstępują wyraźnie od pozostałych rezultatów. Spowodowane jest to tym, że podczas tworzenia nowej sesji połączeniowej *Hibernate* musi załadować bardzo dużą liczbę klas i bibliotek. Sama operacja wykonuje się dosyć szybko, co potwierdzają wyniki testów przeprowadzanych w ramach jednego połączenia. Narzut czasowy generowany jest przez tworzenie połączenia z bazą. Proporcje w różnicach pomiędzy wykonaniem operacji w transakcjach o poziomach izolacji ustawionych na READ_UNCOMMITED i SERIALIZABLE są podobne do tych zanotowanych podczas testów operacji wykonywanych w ramach jednego połączenia. Wykresy przedstawione na rys. 5 i 6 są prawie identyczne, co sugeruje, że poziom izolacji ma znikomy wpływ na czas wykonania transakcji w porównaniu z czasem nawiązywania połączenia z bazą danych.

5. Uwagi końcowe

Niestety, ze względu na wybraną bazę danych i błędy zawarte w wykorzystywanym do badań sterowniku JDBC, część testów nie dała miarodajnych wyników – dotyczy to w szczególności testów procedur składowanych umieszczanych po stronie serwera bazy danych. Można stąd wysnuć wnioski, że łączenie ze sobą produktów firmy Microsoft z technologiami Javy nie jest rozwiązaniem gwarantującym najlepsze wyniki, jeżeli chodzi o wydajność i łatwość tworzenia aplikacji.

Mechanizm zaimplementowany na podstawie specyfikacji EJB, który w testach wypadł w większości przypadków najlepiej, wymaga od projektanta i programisty największego nakładu pracy i najlepszej jej znajomości. Duża liczba interfejsów i metod pomocniczych, które należy zaimplementować skutecznie, odstrasza programistów i czyni tę technologię bardzo nieprzyjemną w praktycznym zastosowaniu pomimo wielu zalet, jakie niesie z sobą każdy serwer aplikacji implementujący EJB.

Biblioteka *Hibernate*, zdecydowanie najłatwiejsza w użyciu, okazuje się technologią bardzo mało wydajną. Pomijając test zapytań realizowanych w jednym połączeniu, otrzymane dla niej wyniki we wszystkich pozostałych testach wychodzą w porównaniach gorzej niż dla innych technologii. Mimo to *Hibernate* jest technologią bardzo popularną, co wynika z bardzo przyjaznego interfejsu dla programisty. Dostępne są m.in. specjalne narzędzia, które na podstawie połączenia z bazą danych generują cały kod odpowiedzialny za komunikację oraz za obsługę błędów.

Interfejs JDBC jest stosunkowo prosty do nauki, przez to chętnie przez początkujących programistów wykorzystywany. Wymaga jednak bardzo sumiennej obsługi błędów, co większości z nich często sprawia kłopoty. Brak dodatkowych udogodnień zapewnianych przez

kontenery uruchomieniowe (jak to ma miejsce w pozostałych omawianych technologiach) powoduje, że wyniki testów dla JDBC okazują się często najmniej korzystnie. Co więcej, w przypadku wielu jednoczesnych połączeń - serwer częstokroć wiele z nich odrzuca.

Podsumowując zatem, można śmiało stwierdzić, że nie ma technologii idealnej, bo każda z badanych ma swoje zalety i wady, i zależnie od wymagań, jakie są stawiane projektowi programistycznemu, spisuje się w niektórych sytuacjach lepiej lub gorzej.

BIBLIOGRAFIA

1. Dobosz K. (red.): Laboratorium programowania w języku Java. Wydawnictwo Politechniki Śląskiej, Gliwice, 2004.
2. Sieradzki R.: Porównanie efektywności dostępu do baz danych w wybranych mechanizmach opartych o maszynę wirtualną Javy. Praca dyplomowa magisterska, Instytut Informatyki, Politechnika Śląska, Gliwice 2007.
3. Bales D.: JDBC. Leksykon kieszonkowy. Helion, Gliwice 2003.
4. Grochala M.: Java - aplikacje bazodanowe. Helion, Gliwice 2001.
5. Roman E., Ambler S., Jewell T.: Enterprise Java Beans. Helion, Gliwice 2003.
6. Platforma Java EE. Witryna internetowa: http://java.sun.com/javaee/.
7. Bauer Ch., King G.: Hibernate w akcji. Helion, Gliwice 2007.

Recenzent: Dr inż. Maciej Bargielski

Wpłynęło do Redakcji 24 lutego 2009 r.

Abstract

The paper presents results of researches on the field of database access with chosen mechanisms using Java Virtual Machine. In the first chapter we can find description of three different technologies: direct access by the JDBC driver, automatic object-relational mapping, and using Enterprise JavaBean objects. The second chapter contains the description of the *JBoss* server used during EJB tests and *Hibernate* tests. Another tool used during tests was MS SQL 2000 database server and its demonstration set of data. The third chapter specifies criteria of tests and tested queries. Four different queries were tested: simple with many

rows in answer, simple with one row in answer, complex with many rows in answer, the most complex with a large pack of rows as a result.

Next the paper concentrates on tests and their results. All of them are analysed and commented. Two first tests realised sending many queries without a transaction: all in one connection and then in separated connections. The best results were getting for Hibernate and EJB CMP. They are illustrated on fig. 1 and fig. 2. After that, tests of transactions were done using two levels of isolation: READ_UNCOMMITED and SERIALIZABLE. After tests of queries done all in one connection (but separated transaction) also Hibernate and EJB CMP gave the best results. Tests of JDBC were failed. After testing queries in separated connections we can that EJB CMP was the best mechanism (or the best implemented) for data persistence. So we get the winner, but we still should remember that nothing is perfect. EJB CMP mechanism (tested version was 2.1) is very complex and hard to use for beginners.

Adresy

Krzysztof DOBOSZ: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, krzysztof.dobosz@polsl.pl
Roman SIERADZKI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, romansieradzki@o2.pl