Optymalizacja warstwy dostępu do danych w aplikacjach korzystających z odwzorowań obiektowo-relacyjnych

Autoreferat pracy doktorskiej
styczeń 2015

Aleksandra Boniewicz

Wydział Matematyki i Informatyki
Uniwersytet Mikołaja Kopernika, ul. Chopina 12/18, 87-100 Toruń

W większości powstających dziś aplikacji za przechowywanie i dostarczanie danych odpowiada baza danych. Nadal najbardziej popularnymi bazami danych są relacyjne bazy danych [14], natomiast najczęściej wybieranymi językami do tworzenia aplikacji są języki obiektowe [16]. Model relacyjny oraz model obiektowy są dwoma zupełnie różnymi światami [6].

Model relacyjny stworzony przez E.F. Codda [7] jest oparty na logice pierwszego rzędu oraz teorii zbiorów. Informacja w tym modelu jest opisana przez predykaty. Relacyjny model danych obejmuje relacje, atrybuty, wartości oraz zmienne relacyjne. Relacja jest pojęciem w sensie teorii zbiorów.

Z drugiej strony model obiektowy jest próbą usystematyzowania informacji w byty zwane obiektami.

Problemy związane z połączeniem świata relacyjnego i obiektowego określa się terminem wziętym z elektrotechniki. Jest nim *niezgodność impedancji* (ang. *impedance mismatch* [17]) Wyróżniamy pięć podstawowych aspektów niezgodności impedancji. Są to [6,9]:

Identyczność obiektów w świecie relacyjnym sprowadza się do identyczności bytów mających ten sam stan. W relacyjnej bazie danych dwie krotki o tych samych wartościach są nierozróżnialne, podczas, gdy dwa obiekty mające identyczny stan są rozróżnialne.

Dziedziczenie nie występuje w modelu relacyjnym.

Stan w modelu relacyjnym jest składowany w tabelach. Aplikacje operują jedynie na tymczasowych kopiach, które mogą stać się nieaktualne, jeśli w międzyczasie zostanie wykonana transakcja zmieniająca stan bazy danych. Nawet jeśli aplikacja skończy działanie, stan bazy danych nie zostanie usunięty. W modelu obiektowym stan obiektów zależy od wykonywanej na nim aplikacji. Gdy aplikacja się kończy, stan obiektu również przestaje istnieć.

Zachowanie W odróżnieniu od relacyjnego modelu danych w modelu obiektowym nie można bezpośrednio manipulować stanem obiektu. Można to robić jedynie poprzez metody. Zbiór operacji na obiekcie nazywany jest interfejsem lub zachowaniem obiektu.

Te zagadnienia dały początek badaniom nad technikami odwzorowania obiektów i relacji [12,11]. Narzędzia *odwzorowania obiektowo-relacyjnego* (ang. *Object/Rational Mapping*, *ORM*) dostarczają obiektowym systemom oprogramowania mechanizmów do przechowywania trwałych danych obiektów w bazie danych z pełną kontrolą transakcyjności. Jednocześnie obiekty w aplikacji [6] pozostają obiektami w rozumieniu języka programowania. Pierwszym tego typu narzędziem był Hibernate stworzony przez Gavina Kinga w 2002 roku [1]. Obecnie większość popularnych języków obiektowych ma narzędzia ORM, np. Hibernate dla Javy, LinQ i ADO dla .NET, SQLObject dla języka Phyton [18].

Narzędzia ORM zapewniają wygodną obsługę automatycznej trwałości obiektów biznesowych aplikacji. Pozwalają programistom skupić się na pisaniu kodu aplikacji a nie na zaawansowanych niuansach języka SQL. Na podstawie metadanych ORM generuje zapytania w dialekcie języka SQL odpowiednim dla aktualnie stosowanej bazy danych. Umożliwia podstawowe operacje CRUD na obiektach klasy zapewniając trwałość i poprawność przetwarzania transakcji. Systemy ORM nie są związane z dostawcami baz danych. Uniezależniają więc programistów od znajomości odpowiedniego dialektu SQL. Zwiększa to przenośność aplikacji. Pozwala wykorzystać podejście, w którym programiści w fazie budowy i testowania aplikacji używają lekkich, lokalnych baz danych. Wdrażają natomiast system działający na innej, bardziej złożonej bazie danych [1]. Narzędzia ORM przyczyniają się również do większej czytelności tworzonej aplikacji. To z kolei ułatwia jej konserwację. Narzędzia te są cały czas rozwijane. Twórcy ORM skupili się jednak głównie na podstawowych problemach trwałości danych. Systemy zarządzania bazą danych (SZBD) oferują natomiast wiele skomplikowanych funkcji i możliwości optymalizacji przetwarzania zapytań realizowanych po stronie bazy danych [2]. Użycie tych funkcjonalności w projektach opartych o ORM rujnuje fundamentalną architekturę proponowaną w ORM. Korzystanie z zaawansowanych funkcji SZBD wymaga bowiem bezpośredniego (z pominięciem warstwy ORM) kontaktu z serwerem bazy danych. Funkcje te w znaczący sposób ułatwiają jednak pracę programistom tworzącym aplikacje obiektowe.

Warto pamiętać, że warstwa odwzorowania obiektowo-relacyjnego stanowi dodatkową abstrakcję składowiska danych. Można ją więc wykorzystać do przezroczystej integracji funkcji SZBD w celu udostępnienia

ich programistom aplikacyjnym. To nie tylko ułatwi pracę programistów, ale również poprawi wydajność aplikacji. Programiści nie będą bowiem musieli stosować brzydkich obejść pozwalających skorzystać z niedostępnych w warstwie ORM, ale bardzo potrzebnych im funkcji SZBD. Użycie tych funkcji z pominięciem interfejsu ORM powoduje nieczytelność kodu, wykonywanie nadmiarowych operacji i wysyłanie zbędnych zapytań do systemu bazy danych.

Celem prezentowanej rozprawy doktorskiej jest zbadanie możliwości zaoferowania zaawansowanych funkcji systemów zarządzania bazami danych na poziomie warstwy odwzorowania obiektowo-relacyjnego. Rozważane są transakcyjne bazy danych. Proponowane rozwiązania powinny być zgodne lub rozszerzać standard JPA (Java Persistence API). Cel ten został osiągnięty. Wykazaliśmy też, że ORM może realizować odpowiednią funkcjonalność nawet wtedy, gdy nie ma jej w użytkowanym SZBD. Skupiliśmy się na zbadaniu jednej funkcjonalności: wykonywaniu zapytań rekurencyjnych. Dla wybranej funkcjonalności przedstawiliśmy jej założenia, prototypową implementację oraz wyniki testów wydajnościowych. Ponieważ nie istnieją aktualnie żadne biblioteki czy narzędzia wspierające proponowaną funkcjonalność dla istniejących narzędzi ORM do budowy prototypu wybrano Hibernate, tj. jedno z najpopularniejszych narzędzi ORM dla obiektowego języka programowania - Javy. Wybór ten był podyktowany tym, że jest to po pierwsze najpopularniejsze narzędzie ORM, a po drugie ma posłużyć jedynie do implementacji i testów wydajnościowych naszej propozycji. Sposoby realizacji oraz rozszerzenia standardu JPA wybranej funkcjonalności (zależnie od możliwości użytkowanego SZBD) opisano w kolejnych rozdziałach.

W rozdziale 3 został przedstawiony benchmark służący do pomiaru wydajności implementacji zapytań rekurencyjnych zgodnych ze standardem SQL:1999. Istnieje wiele firm oraz organizacji, które definiują standardy benchmarków, m.in. SPEC (*Standard Performance Evaluation Cooperation*) [19]. W dziedzinie baz danych najpopularniejsze benchmarki definiuje konsorcjum TPC (*Transaction Processing Performance Council*) [8,10]. Niestety żaden z benchmarków dla relacyjnych baz danych (także tych definiowanych przez TPC) nie uwzględnia zapytań rekurencyjnych wg standardu SQL:1999.

W ramach prac nad niniejszą rozprawą został przygotowany odpowiedni benchmark. Jego celem jest porównanie implementacji zapytań rekurencyjnych SQL:1999 a także alternatywnych rozwiązań zaproponowanych w niniejszej rozprawie doktorskiej. Pierwszą wersję tego bench-

marku wraz z wynikami jego ewaluacji dla wybranych komercyjnych systemów zarządzania bazami danych przedstawiono w [13].

W rozdziale 4 rozprawy zostało opisane rozszerzenie narzędzi odwzorowań obiektowo-relacyjnych o obsługę zapytań rekurencyjnych zgodnych ze standardem SQL:1999. Wiele systemów baz danych wspiera zapytania rekurencyjne [13]. W rozdziale 4 założyliśmy taką sytuację, tzn. użytkowany SZBD ma zaimplementowane zapytanie rekurencyjne SQL:1999. Chociaż w narzędziach typu ORM można wyniki takich zapytań obliczyć iteracyjnie po stronie aplikacji, jednak taki sposób ich wykonania nie jest optymalny. Opracowaliśmy więc rozszerzenie narzędzi ORM o wydajne mechanizmy obsługi zapytań rekurencyjnych składające się z odpowiedniego interfejsu programisty aplikacyjnego oraz procedur odwzorowania zapytań rekurencyjnych aplikacji na zapytania zapisane w dialekcie SQL użytkowanego SZBD. Interfejs programistyczny będący przedmiotem rozdziału 4 po raz pierwszy zaprezentowano w artykule [18]. Rozszerzenie to zostało prototypowo zaimplementowane i przetestowane w Hibernate. Szczegóły techniczne tej prototypowej implementacji można znaleźć w pracy magisterskiej [15]. Wykonane testy pokazują, że czas wykonania zapytań rekurencyjnych zaproponowany w tym rozszerzeniu jest znacznie szybszy od metod natywnych dostępnych w Hibernate. Wynika to z faktu, że dla metod natywnych do bazy danych było wysyłanych tyle zapytań, ile obiektów ma dana struktura.

W rozdziale 5 rozprawy przedstawiono i zanalizowano mechanizmy realizacji zapytań rekurencyjnych w narzędziach ORM dla tych SZBD, które nie implementują zapytań rekurencyjnych (np. bardzo popularny MySQL). Mechanizmy te zostały opisane w artykule [3]. Mechanizmy te uproszczą pracę programistów, którzy w przypadku takich systemów baz danych muszą zapytania rekurencyjne wykonywać w sposób iteracyjny niezależnie od tego czy używają ORM, czy nie. Obciąża to bazę danych i sieć oraz komplikuje kod aplikacji. Niestety ta najbardziej intuicyjna metoda jest jednak też najbardziej kosztowną. W rozdziale 5 przedstawiliśmy trzy metody realizacji zapytań rekurencyjnych bez odwoływania się do składni SQL:1999: iterację bezpośrednią (*direct loop*), rozwinięcie w głąb (ang. *vertical unrolling*) oraz rozwinięcie wszerz (ang. *horizontal unrolling*). Iteracja bezpośrednia służyła jako rozwiązanie referencyjne odpowiadające istniejącym możliwościom realizacji zapytań rekurencyjnych po stronie klienta. Metoda rozwinięcia wszerz ma zastosowanie, jeśli znamy głębokość danych rekurencyjnych lub chcemy ograniczyć głębokość przeszukiwania. Zamiast wysyłać wiele pojedynczych zapytań do bazy danych, budujemy jedno zapytanie złożone z samozłączeń lewostronnych

tej samej tabeli. W metodzie rozwinięcia w głąb zamiast dodawać nowe kolumny (jak w rozwinięciu wszerz) dodajemy nowe wiersze uzyskiwane poprzez kolejne sumy mnogościowe z coraz głębszymi samozłączeniami. Otrzymujemy więc zapytanie, które jest sumą mnogościową złączeń tabeli samej ze sobą odpowiednio tyle razy, ile jest poziomów zagłębienia. Wszystkie metody zaimplementowano i przetestowano ich wydajność na danych o różnych wielkościach i głębokościach hierarchii. Z przeprowadzonych testów wynika, że metody rozwijające zapytania rekurencyjne (czy to w głąb czy wszerz) przedstawione rozdziale 5 rozprawy są istotnie szybsze niż iteracja bezpośrednia. Z tych dwóch metod szybszym okazało się rozwinięcie wszerz. Jest ono nawet czterokrotnie szybsze od rozwinięcia w głąb dla dwudziestu poziomów zagłębienia danych. Duża liczba kolumn w wyniku nie stanowi dla tej metody przeszkody.

W rozdziale 6 rozprawy omówiono cztery odmienne mechanizmy realizacji zapytań rekurencyjnych. Zasadnicza różnica między metodami opisanymi w rozdziale 5, a tymi z rozdziału 6, polega na użyciu zmaterializowanych danych redundantnych. Metody z rozdziału 6 mogły takie dane gromadzić i wykorzystywać. Metody takie można stosować zarówno wtedy gdy używany SZBD realizuje zapytania rekurencyjne SQL:1999, jak i wtedy, gdy ich nie realizuje. Metody te opisano w artykułach [4] oraz [5]. Są to: metoda ścieżek pełnych, metoda ścieżek logarytmicznych, metoda zbiorów zagnieżdżonych i metoda ścieżek zmaterializowanych. Wszystkie zaimplementowano w prototypowym rozszerzeniu Hibernate ORM. Następnie przetestowano tę implementację, a jej wyniki umieszczono w rozdziale 6 rozprawy. Okazało się, że w określonych warunkach każda z tych czterech metod może okazać się lepsza od pozostałych oraz od tych niekorzystających z danych redundantnych. W wyniku przeprowadzonych badań ustaliliśmy też, kiedy poszczególne sposoby odwzorowania zapytań rekurencyjnych aplikacji mogą okazać się przydatne w konkretnych sytuacjach w praktyce.

Dodatkowo został podjęta próba zdefiniowana analitycznej funkcji kosztu dla zadanego obciążenia (ang. *workload*) bazy danych zapytaniami rekurencyjnymi i operacjami modyfikacji danych. Taka funkcja może być stosowana w celu wskazania najlepszej metody materializacji ścieżek w celu wsparcia zapytań rekurencyjnych, gdy stosowany system zarządzania bazą danych ich nie implementuje. Budując taką funkcję należy wziąć pod uwagę zapytania oraz operacje modyfikacji danych. Te ostatnie mogą stanowić istotny ułamek kosztów obciążenia bazy danych, ponieważ obejmują nie tylko modyfikację danych bazowych, ale także odpowiednią pielęgnację zmaterializowanych danych redundantnych. Dla każdej z czterech

metod opisanych w tym rozdziale została zaproponowana funkcja kosztu realizacji obciążenia złożonego z zapytań i modyfikacji. Funkcja ta zależy od parametrów takich jak liczba zapytań rekurencyjnych, liczba operacji modyfikacji danych rekurencyjnych, selektywność zapytań, liczba rekordów oraz głębokość hierarchii.

Celem rozdziałów 4–6 prezentowanej rozprawy doktorskiej jest zbadanie możliwości i zasadności rozszerzania systemów odwzorowań obiektowo-relacyjnych o udogodnienia do wykonywania zapytań rekurencyjnych aplikacji. Okazało się, że jest to możliwe oraz że tego typu udogodnienia mogą zwiększyć wydajność aplikacji i czytelność jej kodu źródłowego niezależnie od tego, czy użytkowany system zarządzania bazami danych implementuje zapytania rekurencyjne SQL:1999, czy nie.

Literatura

1. C. Bauer, G. King. *Hibernate w akcji.* Wydawnictwo Helion, 2007.
2. A. Boniewicz, M. Gawarkiewicz, P. Wiśniewski. Automatic selection of functional indexes for object relational mapping system. *International Journal of Software Engineering and Its Applications*, 7(4), 2013.
3. A. Boniewicz, K. Stencel, P. Wiśniewski. Unrolling SQL:1999 recursive quries. T.-H. Kim, J. Ma, W.-C. Fang, Y. Zhang, A. Cuzzocrea, redaktorzy, *Computer Applications for Database, Education, and Ubiquitous Computing*, wolumen 352 serii *Communications in Computer and Information Science*, strony 345–354. Springer, 2013.
4. A. Boniewicz, K. Stencel, P. Wiśniewski. On materializing paths for faster recursive querying. B. Catania, T. Cerquitelli, S. Chiusano, G. Guerrini, M. Kämpf, A. Kemper, B. Novikov, T. Palpanas, J. Pokorný, A. Vakali, redaktorzy, *New Trends in Databases and Information Systems*, wolumen 241 serii *Advances in Intelligent Systems and Computing*, strony 105–112. Springer, 2014.
5. A. Boniewicz, P. Wiśniewski, K. Stencel. On redundant data for faster recursive querying via ORM systems. M. Ganzha, L. A. Maciaszek, M. Paprzycki, redaktorzy, *FedCSIS*, strony 1439–1446, 2013.
6. T. Broek. Object relational mappings, jan 2007.
7. E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.
8. T. P. P. Council. TPC Benchmarks. Online, 2009.
9. M. L. Fussel. Foundation of object relational mappings, jan 1997.
10. J. Gray, redaktor. *The Benchmark Handbook for Database and Transaction Systems (2nd Edition).* Morgan Kaufmann, 1993.
11. W. Keller. Mapping objects to tables - a pattern language. *Proc. Of European Conference on Pattern Languages of Programming Conference (EuroPLOP)"97*, 1997.
12. S. Melnik, A. Adya, P. A. Bernstein. Compiling mappings to bridge applications and databases. *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, strony 461–472, New York, NY, USA, 2007. ACM.

13. P. Przymus, A. Boniewicz, M. Burzańska, K. Stencel. Recursive query facilities in relational databases: A survey. Y. Zhang, A. Cuzzocrea, J. Ma, K.-I. Chung, T. Arslan, X. Song, redaktorzy, *FGIT-DTA/BSBT*, wolumen 118 serii *Communications in Computer and Information Science*, strony 89–99. Springer, 2010.
14. Solid IT. DB-Engines ranking. Online, 2013.
15. A. Szumowska. Wsparcie dla zapytań rekurencyjnych w odwzorowaniach obiektowo-relacyjnych. Praca magisterska, Wydział Matematyki i Informatyki, Uniwersytet Mikołaja Kopernika w Toruniu, 2011.
16. Tiobe Software. Programming community index. Online, 2013.
17. P. Wiśniewski, M. Burzańska, K. Stencel. The impedance mismatch in light of the unified state model. *Fundam. Inform.*, 120(3-4):359–374, 2012.
18. P. Wiśniewski, A. Szumowska, M. Burzańska, A. Boniewicz. Hibernate the recursive queries - defining the recursive queries using Hibernate ORM. J. Eder, M. Bieliková, A. M. Tjoa, redaktorzy, *ADBIS (2)*, wolumen 789 serii *CEUR Workshop Proceedings*, strony 190–199. CEUR-WS.org, 2011.
19. M. Wojciechowski, M. Zakrzewicz. TPC Benchmarking. *Materiały VIII konf. PLOUG*, Systemy informatyczne: projektowanie, implementowanie, eksploatowanie, strony 244–259. Stowarzyszenie Polskiej Grupy Użytkowników systemu Oracle, oct 2002.