Implementacja języka zapytań oparta na abstrakcyjnych drzewach składniowych

(Implementation of a query language
based on abstract syntax trees)

Damian Górski

Praca inżynierska

Promotor: dr Wiktor Zychla

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

30 czerwca 2017 r.

Damian Górski

.....

.....
(adres zameldowania)

.....

.....
(adres korespondencyjny)

PESEL: .....

e-mail: .....

Wydział Matematyki i Informatyki
stacjonarne studia I stopnia
kierunek: informatyka
nr albumu: 273212

Oświadczenie
o autorskim wykonaniu pracy dyplomowej

Niniejszym oświadczam, że złożoną do oceny pracę zatytułowaną *Implementacja języka zapytań oparta na abstrakcyjnych drzewach składniowych* wykonałem/am samodzielnie pod kierunkiem promotora, dr Wiktora Zychli. Oświadczam, że powyższe dane są zgodne ze stanem faktycznym i znane mi są przepisy ustawy z dn. 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tekst jednolity: Dz. U. z 2006 r. nr 90, poz. 637, z późniejszymi zmianami) oraz że treść pracy dyplomowej przedstawionej do obrony, zawarta na przekazanym nośniku elektronicznym, jest identyczna z jej wersją drukowaną.

Wrocław, 30 czerwca 2017 r.

(czytelny podpis)

Streszczenie

Gdy projektujemy pewien system informatyczny, zazwyczaj musimy zmierzyć się z wyborem bazy danych, którą chcemy użyć w naszej aplikacji. Problem pojawia się w momencie, gdy pewna baza danych oferuje to, czego szukamy, ale potrzebujemy sposobu wybierania z niej informacji w wygodny dla nas sposób. W technologii .NET pozwala na to Language INtegrated Query (LINQ), który tłumaczy zapytanie w swoim języku na abstrakcyjne drzewo składniowe, po którym można przejść, implementując pewien zbiór obiektów odwiedzających (visitorów), w celu stworzenia zapytania w bazie danych naszego wyboru. Tematem tej pracy jest implementacja takich odwiedzających, którzy zbudują zapytanie do bazy PostgreSQL.

While projecting a computer system, we frequently have to cope with the task of choosing the database we want to use in our application. The problem is, some databases offer what we need, but we also need a more comfortable way to obtain information from it. The .NET framework allows us to achieve that with Language INtegrated Query (LINQ), which translates a query in its language into an abstract syntax tree, that we can traverse by implementing a set of visitors in order to create a query in our target database. The topic of this thesis is implementing such visitors that build a query to the PostgreSQL database.

Spis treści

Rozdział 1.

Preliminaria

Aby zrozumieć mechanizm budowy zapytania SQL-owego, trzeba zrozumieć, jakie struktury danych są przez niego wykorzystywane. Zakładam, że czytelnikowi znane są podstawowe pojęcia związane z programowaniem obiektowym, takie jak metoda, kolekcja, dziedziczenie, typ generyczny. W tym rozdziale poruszone zostaną następujące tematy:

- Sposób przetrzymywania wyliczalnych kolekcji w .NET-cie.
- Opis technologii LINQ, która jest punktem wejścia dla zapytania użytkownika.
- Struktura drzewa wyrażeń `IQueryable`, i dlaczego takie drzewa są trudne.
- Biblioteka re-linq, obiekty `QueryModel`.

1.1. Słowo o `IEnumerable<T>` i `IQueryable<T>`

We frameworku .NET wszystkie kolekcje, które możemy wyliczyć (a takie nas interesują, bo pracujemy z relacyjną bazą danych), implementują interfejs `IEnumerable<T>`, gdzie `T` jest typem obiektu, który jest przetrzymywany w kolekcji. Ten interfejs definiuje metodę `GetEnumerator()`, który zwraca obiekt typu `IEnumerator<T>`, który ma właściwość `Current` oraz metodę `MoveNext()`, pozwalając na p'rzejście po uporządkowanym ciągu obiektów typu `T` oraz określenie obecnej pozycji. Korzystając z tych dwóch informacji, jesteśmy w stanie rozszerzyć `IEnumerable<T>` o metody takie jak wyznaczenie długości, filtrowanie kolekcji, łączenie dwóch kolekcji ze sobą, mapowanie funkcji na obiekty. Dokładna lista metod rozszerzających `IEnumerable<T>` jest dostępna w oficjalnej dokumentacji MSDN.

Rozszerzeniem `IEnumerable<T>` jest interfejs `IQueryable<T>`, który de facto implementuje `IEnumerable<T>`. Zasadniczą różnicą między tymi dwoma interfejsami jest to, że w momencie wywołania ciągu metod rozszerzających `IEnumerable<T>`,

każda z tych metod jest wywoływana jedna po drugiej, co może obciążyć moc obliczeniową procesora. Natomiast kolekcja `IQueryable<T>` jest świadoma, że nie musi wykonywać tych metod od razu, tylko przetrzymuje je w postaci drzewa wyrażeń (o wyrażeniach w następnej sekcji), które dopiero przy wywołaniu metody wyliczającej elementy z kolekcji zostaje wykonane w całości w efektywny sposób. Takie rozwiązanie jest idealne dla kolekcji, które łączą się z zewnętrzną bazą danych, aby istniała możliwość wybrania danych za pomocą jednego dużego zapytania SQL-owego.

1.2. Drzewa wyrażeń `IQueryable<T>`

todo.

1.3. Language INtegrated Query

todo.

1.4. re-linq i `QueryModel`

Aby można było odwołać się do bazy danych za pomocą LINQ, konieczne jest zaimplementowanie interfejsu `IQueryable` oraz `IQueryProvider`. W sekcji traktującej o drzewach wyrażeń `IQueryable<T>` można zobaczyć, że to może być trudne, ze względu na skomplikowaną strukturę tych drzew. W związku z tym alternatywnym rozwiązaniem jest biblioteka re-linq, która tłumaczy drzewa wyrażeń `IQueryable` na tytułowe abstrakcyjne drzewa składniowe, a dokładniej - na obiekty `QueryModel`, które o wiele bardziej przypominają oryginalne zapytanie LINQ. W `QueryModel` interesują nas cztery właściwości:

- `SelectClause` - zawiera wyrażenie określające element, który jest wybierany w zapytaniu LINQ.
- `MainFromClause` - określa główne źródło, z którego wybierane są informacje (w przypadku zapytań SQL - pierwsza tabela z części `FROM`).
- `BodyClauses` - zawiera kolekcję
- `ResultOperators` -

Następnie, takie drzewo jest przekazywane do jednej z metod implementacji `IQueryExecutor`. Ten interfejs ma trzy metody o sygnaturach:

- `IEnumerable<T> ExecuteCollection<T>(QueryModel queryModel)`,

- `T ExecuteScalar<T>(QueryModel queryModel)`,
- `T ExecuteSingle<T>(QueryModel queryModel, bool defaultWhenEmpty)`.

Te metody są punktami wejścia i wyjścia dla całego procesu budowy zapytania, jego wykonania oraz konwersji wyniku do oczekiwanego formatu i są wywoływane są w zależności tego, jaki format jest oczekiwany (cała kolekcja, skalar, pojedynczy element z kolekcji).

re-linq implementuje wzorzec projektowy Odwiedzający (Visitor), który przechodzi przez zbudowany przez siebie `QueryModel` i pozwala programiście na przeciążenie metod, które przechodzą przez odpowiadające typy wyrażeń, w celu zbudowania zapytania do docelowej bazy danych. Przechodzenie przez zapytania w takiej postaci jest zdecydowanie bardziej przystępne dla programisty, który podejmuje się zadania LINQ-to-SQL, niż żmudne próby budowy zapytania SQL-owego na podstawie drzewa wyrażeń `IQueryable<T>`. W następnym rozdziale poruszona zostanie kwestia przechodzenia przez `QueryModel` w celu zbudowania zapytania SQL-owego.

Warto jeszcze zaznaczyć, że biblioteka re-linq jest na tyle potężnym narzędziem, że na jej użycie zdecydowali się nawet autorzy NHibernate oraz Entity Framework 7, które są najpopularniejszymi bibliotekami ORM w .NET.

Rozdział 2.

Proces budowy zapytania

Tutaj coś będzie jak złapię wenę.

Rozdział 3.

Testy jakości i wydajności

Tutaj coś będzie jak złapię wenę.

Rozdział 4.

Podsumowanie

Tutaj coś będzie jak złapię wenę.

Bibliografia

Dodatek A

Instrukcja obsługi

Tutaj coś będzie jak złapię wenę.