## Unit of Work i Repozytorium

W tym rozdziale zostanie omówiona kwestia dostępu do danych przetrzymywanych bezpośrednio w naszej bazie danych. Zostanie poruszona kwestia wzorców projektowych Unit of Work i Repozytorium oraz porównanie ich zastosowania.

Zanim przejdziemy do szczegółów implementacji samego wzorca, przyjrzyjmy się technologiom pozwalającym nam na odczyt i zapis danych do bazy. W omawianej aplikacji zrezygnowano z ręcznego tworzenia zapytań sql na rzecz technologii ORM. ORM, czyli Object Relational Mapping to sposób odwzorowania relacyjnej bazy danych lub innej relacyjnej struktury na obiektową architekturę. Jako silnika ORM został użyty Entity Framework. Bliżej zarówno o Entity Framework jak i samej idei ORM można przeczytać w rozdziałach poświęconych wykorzystanym technologiom. Tutaj skupimy się na praktycznym wykorzystaniu powyższych technologii przy implementacji omawianych wzorców.

Repozytorium jest wzorcem, który jest pomostem pomiędzy warstwą logiki biznesowej, a modułami zajmującymi się dostępem do danych zapisanych w bazie danych. Wprowadza ona dodatkową warstwę abstrakcji, dzięki której w łatwiejszy sposób możemy implementować operacje oparte o zapytania do bazy. Dzięki niemu jesteśmy wstanie odprawować zapytania SQL bądź technologię ORM od naszego warstw wyższych, zasilanych danymi pobranymi z bazy danych. Przyjmowane jest założenie, że każda implementacja repozytorium dedykowane jest konkretnej relacji tabela – obiektowy model danych. Wszystkie repozytoria implementują wspólny interfejs IRepository, w którym zadeklarowane są podstawowe metody do obsługi danych – dodawanie, usuwanie, edycja oraz pobieranie. Nic nie stoi na przeszkodzie, aby każda konkretna implementacja repozytorium była rozszerzona o dodatkowe metody, specyficzne dla wybranego typu danych.

Stosowanie powyższego wzorca niesie za sobą jednak pewne konsekwencje. W sytuacji, gdy stosujemy technologię ORM, część jej właściwości jest zdublowana przez repozytorium. Ponadto niektóre z metod i właściwości zapewnianych przez silnik ORM jest niedostępna, bez ich zaimplementowania w repozytorium. Może być to sytuacja pożądana, jeżeli chcemy, aby pozostałe warstwy miały dostęp tylko do wybranych przez nas funkcji. Czasami prowadzi to jednak do dużej ilości nadmiarowego kodu. Kolejnym problem w pracy z repozytoriami w klasycznym tego słowa znaczeniu jest tworzenie transakcji. Spowodowane jest to tym, że każda tabela posiada własną implementację repozytorium, a co za tym idzie, każde repozytorium posiada osobną metodę SaveChanges. Jeżeli dana operacja przeprowadzana jest tylko na jednej tabeli, to żaden problem nie występuje. Po prawnym wprowadzeniu zmian, możemy je bezproblemowo zapisać. Jednak zdecydowana większość operacji przeprowadzana jest równocześnie na większej ilości tabel. Powoduje to, że to programiście spoczywa uruchomienie funkcji SaveChanges w odpowiednim momencie. Brak zapisu zmian którymś z używanych repozytorium może powodować niespójność danych, nad którymi pracujemy.

Jak widać wzorzec Repozytorium ma swoje wady i zalety. Jego główne atuty widzimy wtedy, jeżeli używamy bezpośredniego dostępu do bazy danych z użyciem kwerend SQL. Następuje wtedy enkapsulacja kodu SQL pod postacią implementacji konkretnych metod Repozytorium. Postać tych zapytań różni się, w zależności od tabeli, która jest źródłem danych dla danej implementacji. Posiadamy wtedy zbiór klas implementujących metody zawarte w danym interfejsie, jednak robiących to na różne sposoby. Nieco inaczej sytuacja wygląda jednak w przypadku użycia Frameworków ORM. Jak już wiemy, w naszej aplikacji został zastosowany Entity Framework. Przy jego pomocy uniknęliśmy konieczności pisania kwerend. Cała obsługa bazy danych tworzona jest w C# oraz bazuje na modelu obiektowym. Wszystkie operacje przeprowadzone na dowolnej tabeli, wykonywane według tego samego schematu, różni się, tylko model, na którym pracujemy. W zaistniałej sytuacji tworzenie dodatkowych repozytoriów, enkapsulujących operacje na danym modelu danych (danej tabeli) może być tworzeniem nadmiarowej abstrakcji.

Alternatywą lub rozszerzeniem dla wzorca Repozytorium, jest wzorzec projektowy Unit Of Work, a w szczególności jego generyczna implementacja. Unit Of Work w założeniu daje dostęp operacji na dowolnej tabeli, jaka zostanie pod niego podpięta. Zasilany może być zarówno przez repozytoria, które dostarczają mu danych do pracy na konkretnych tabelach, jak i przez silnik ORM.

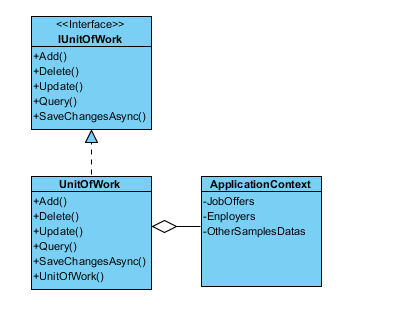
Zanim przejdziemy do omówienie implementacji samego Unit of Work, przyjrzyjmy się klasie ApplicationContext.



Jak widać, klasa ta dziedziczy po klasie DbContext, która wchodzi w skład biblioteki EntityFramework. Jednym ze skutków takiego dziedziczenia jest możliwość tworzenia generycznych DbSetów. Każdy DbSet przyjmuje jako generyczny parametr strukturę danych. Na podstawie tych struktur oraz relacji między nimi, może zostać wygenerowana odpowiadająca im baza danych. Z drugiej strony istnieje możliwość wygenerowania odpowiedniej implementacji DbConext na podstawie istniejącej bazy danych. Z perspektywy wzorca UoW istotną właściwością DbSetów, są metody, które umożliwiają dodawanie, usuwanie, edycje oraz pobieranie odpowiadających. Implementują więc wszystkie podstawowe funkcje Repozytoriów. Spójrzmy teraz na interfejs IUnitOfWork oraz implementującą go klasę.





Jak widać na powyższych listingach w interfejsie IUnitOfWork zadeklarowane są metody CRUD (Create, Read, Update, Delete). Metody są generyczne, dzięki klauzuli where parametrem danej metody może dowolna klasa. W klasie implementującej ten interfejs możemy się przyglądnąć ciałom tych metod. Każda z nich wykorzystuje na swój sposób omówioną wcześniej klasę ApplicationContext. Operacje dodawania, usuwania i pobierania, korzystają z metody Set, która dzięki generycznemu parametrowi odwołuje się do wcześniej zdefiniowanego DbSetu. Dzięki temu, mogą one przyjąć dowolny typ danych, jaki został wcześniej zarejestrowany w ApplicationContext. Na podobnej zasadzie działa operacja Update, która wywołuje generyczną metodę Entry. Istotny jest tutaj fakt, że parametrem, z którym ta funkcja zostanie wywołana, jest wcześniej pobrany i przemapowany do obiektu, rekord z bazy danych. Ostatnią funkcją zawartą w UoW jest SaveChangesAsync. Służy ona do faktycznego zapisania zmian w bazie danych. To jej wywołanie zapisuje zmiany wprowadzone przez pozostałe funkcje. Zwróćmy jeszcze uwagę, że wszystkie metody zwracają Task. Dzięki czemu mogą być uruchomiane równolegle na osobnych wątkach, co znacznie zwiększa wydajność.