Fundusze Europejskie
Wiedza Edukacja Rozwój

Politechnika Wrocławska

Unia Europejska
Europejski Fundusz Społeczny

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Politechnika Wrocławska

Wydział Informatyki i Zarządzania

Zaawansowane Technologie Webowe

Laboratorium

Temat:	Projekt grupowy – Migracja bazy danych
Opracował:	mgr inż. Piotr Jóźwiak
Data:	lipiec 2020
Liczba godzin:	2 godziny

Table of Contents

Wstęp

W czasie, gdy GIT zdobywał swoją popularność, powstał trend do implementowania aplikacji webowych z wykorzystaniem mapowania obiektowo-relacyjnego (*object relational mapping* – ORM). Główną ideą powstania tej funkcjonalności było zauważenie, że skoro programiści wprowadzają zmiany w kodzie, które z łatwością można wycofać za pomocą Git, to podobne rozwiązanie należy zastosować w schemacie bazy danych. W końcu każda większa nowa funkcjonalność obejmuje zmiany kodu i schematu jednocześnie. Tak popularne narzędzia jak **Rails** i **Django** dodały ORM i migrację bazy danych (znaną również jako migracja schematu) jako część frameworka.

Jednak koncepcja migracji bazy danych nie ogranicza się do popularnych platform internetowych. Istnieją niezależne biblioteki do migracji baz danych, takie jak Flyway (https://flywaydb.org/) i Liquibase (https://www.liquibase.org/). Wyobraźmy sobie możliwość cofania szczegółowych zmian w schemacie podczas pisania kodu.

Jak działa migracja bazy danych

Skoro wiemy skąd pojawiła się koncepca migracji schemtów to przyjrzyjmy się sposobie funkcjonowania.

Poszczególne zmiany jako skrypty

Wspominaliśmy już, że migracja schematu bazy danych polega w zasadzie na szczegółowym śledzeniu zmian w jej strukturze (czasem także zmian w samych danych). Śledzenie odbywa się poprzez odzwierciedlenie każdej ze zmian jako oddzielne pliki skryptów. W ten sposób każda zmiana zostaje sformalizowana w postaci kodu, który w łatwy sposób poddaje się przechowywaniu w systemie kontroli wersji.

Jako przykład pliku migracji niech posłuży poniższy listing z frameworka Rails:

```
class CreateProducts < ActiveRecord::Migration[5.0]
    def change
      create_table :products do |t|
        t.string :name
        t.text :description

        t.timestamps
      end
    end
  end
```

Jak widać z powyższego przykładu, plik migracji to formalny zapis zmian w schemacie bazy danych. Więcej o działaniu migracji w frameworku Ruby on Rails można znaleźć tutaj: https://www.c-sharpcorner.com/article/create-migration-file-in-ruby-on-rails/. Oczywiście można utworzyć stan na podstawie aktualnego stanu bazy danych, więcej o tym tutaj: https://edgeguides.rubyonrails.org/active_record_migrations.html#schema-dumping-and-you.

Pliki migracji tworzą pełną historię zmian jaką należy wykonać z danego punku w przeszłości, aby osiągnąć aktualną wersję schematu – pasującą do kodu aplikacji (w naszym przypadku webowej).

Migracje z użyciem zewnętrznych narzędzi

W poprzednim przykładzie przedstawiliśmy sposób migracji bazy danych wbudowany we framework Ruby. Tylko co zrobić, jeśli używany przez nas framework nie posiada takiej funkcjonalności? Wtedy do tego celu należy wykorzystać zewnętrzne narzędzie.

Poniższy przykład przedstawia formalizm migracji z Liquibase:

```xml
<?xml version="1.0" encoding="UTF-8"?>

<databaseChangeLog
  xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
         http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.1.xsd">

    <changeSet id="1" author="bob">
        <createTable tableName="department">
            <column name="id" type="int">
                <constraints primaryKey="true" nullable="false"/>
            </column>
            <column name="name" type="varchar(50)">
                <constraints nullable="false"/>
            </column>
            <column name="active" type="boolean" defaultValueBoolean="true"/>
        </createTable>
    </changeSet>

</databaseChangeLog>
```

Tym razem jest on w formacie XML. Liquibase wspiera zapis migracji lub inaczej changeset (zgodnie z tym jak nazywają to w Liquibase) w wielu formatach, np. JSON lub YAML (https://docs.liquibase.com/concepts/basic/changelog.html).

Jeśli więc nie chcemy ręcznie pisać migracji bazy danych w formacie SQL, nie ma żadnych standardów dotyczących sposobu tworzenia plików migracji. Oczywiście w plikach SQL można napisać własne niestandardowe skrypty migracji bazy danych. Ale po co pisać odręcznie i przypadkowo wprowadzać błędy, skoro można użyć narzędzi do automatycznego generowania migracji bazy danych?

Jak przeprowadzić migrację?

Teraz, kiedy wiemy czym jest migracja bazy danych, przyjrzyjmy się (przynajmniej koncepcyjnie), jak ją przeprowadzić. Niestety na tę chwilę nie ma żadnych standardów dotyczących sposobu zapisu migracji, co oznacza, że nie możemy wdać się tutaj w zbytnie szczegóły tworzenia. Innymi słowy, sposób przeprowadzenia migracji bazy danych w dużej mierze zależy od konkretnego narzędzia używanego do tego zadania.

Omówmy dwa najpopularniejsze sposoby przeprowadzania migracji bazy danych.

Wykorzystanie frameworkowych narzędzi i bibliotek

Jeśli używamy popularnego języka (Ruby, PHP, Python, itp.) lub frameworka (Rails, Django, itp.), to istnieją dobrze udokumentowane biblioteki do migracji dostępne wewnątrz frameworka. Funkcjonalność taka często jest dostarczna jako część frameworka. Czasem można spotkać więcej niż jedną bibliotekę dostępną w danym środowisku.

Zazwyczaj w tym scenariuszu pliki migracji są generowane za pomocą wiersza polecenia. Czasami może być konieczne ręczne napisanie kodu niestandardowego w przypadku niektórych zmian, takich jak migracja danych, a nawet sposób cofnięcia samej zmiany. Poza tym, większość żmudnej pracy wykonuje narzędzie do migracji.

Wykorzystanie narzędzi zewnętrznych

W niektórych przypadkach będziemy zmuszeniu (lub chcieli) użyć oprogramowania zewnętrznego, takiego jak Flyway lub Liquibase, które działa na zasadzie kontroli wersji schematu bazy danych. Skorzystanie z tego typu rozwiązania ma dużo zalet. Unika się w ten sposób silnego związania z określonym frameworkiem, a zatem łatwiej będzie można zmienić język programowania – bez zbędnego martwienia się o porting migracji.

Niestety same narzędzia także wprowadzają swego rodzaju ograniczenia. Dla przykładu Flyway wspiera pracę z bazami danych jak Oracle, MySQL czy MaridDB, ale katalog dostępnych baz jest zamknięty. Samo skorzystanie z Flywaya wymaga użycia Javy, co także może okazać się kolejnym ograniczeniem technologicznym.

Dobra wiadomość jest taka, że Flyway i Liquibase są stosunkowo łatwe w użyciu. Umożliwiają one również generowanie migracji z poziomu wiersza polecenia i umożliwiają niestandardowe kodowanie w celu przechwycenia migracji schematu bazy danych.

Które podejście jest najlepsze?

Zatem, które rozwiązanie jest najlepsze? Większość programistów wybiera opcję pierwszą.

Zwykle dzieje się tak dlatego, że programiści mają już preferowany język / strukturę, więc mówiąc kognitywnie, nie wydaje się im, że uczą się (jeszcze) innego oprogramowania. A uzależnienie (w przypadku wariantu pierwszego dotyczyłoby ram i języka) jest całkowicie dobrowolne i pożądane. Opcja druga wydaje się dobrym rozwiązaniem, jeśli zespół nie jest w pełni przywiązany do określonego języka lub struktury.

Niezależnie od tego, na co się zdecydujemy, unikajmy niepotrzebnej zmiany swojego wyboru w połowie rozwoju aplikacji. Migracja bazy danych nie jest obszarem, w którym zmiana narzędzi daje ogromne korzyści. Większość korzyści wynika z prostego skonfigurowania migracji bazy danych w pierwszej kolejności.

Podsumowanie

Migracja bazy danych to jedno z tych niezbędnych narzędzi dla programisty, które znacząco ułatwia pracę. W przyszłości przewiduję, że migracja baz danych będzie ewoluować z najlepszej praktyki programistycznej do standardowej praktyki programistycznej. Mimo to ważne jest, aby pamiętać, jak

migracje baz danych mogą obrócić się przeciwko programiście, szczególnie w przypadku trudnych do odwrócenia zmian schematu. Miejmy absolutną pewność co do zmian, zanim wprowadzimy je w życie.

Dlatego, każdy zespół ma za zadanie przeanalizować dostępne rozwiązania we własnym stosie technologicznym i dobranie najbardziej dopasowanego do zastanej sytuacji. Należy wprowadzić do własnego projektu funkcjonalność migracji bazy danych.