Wybrane elementy praktyki projektowania oprogramowania
Zestaw 8

Bazy danych, SQL

2019-01-08

Liczba punktów do zdobycia: **12/82**
Zestaw ważny do: 2019-01-22

Uwaga! Zadania należy rozwiązać posługując się wybranym przez siebie systemem relacyjnych baz danych, przy czym rekomendacja jest taka, żeby wybrać jeden z dwóch: PostgreSQL lub SQL Server.

1. (**2p**) Utworzyć tabelę `OSOBA` przechowującą podstawowe dane osobowe, imię, nazwisko, płeć, itd. Wybrać odpowiednie typy dla kolumn, uzasadnić wybór. Do roli klucza głównego zdefiniować dodatkową kolumnę typu całkowitoliczbowego (tzw. *surrogate key*).

   W pierwszym wariancie zadania (za 1p) zdefiniować w bazie sekwencję dostarczającą wartości dla klucza głównego. Pokazać jak za pomocą języka SQL wstawiać rekordy do tabeli najpierw pobierając kolejną wartość z sekwencji a następnie używając jej w kwerendzie typu `INSERT INTO`.

   W drugim wariancie zadania (za 1p) nie definiować sekwencji, zamiast tego kolumnę klucza głównego zdefiniować tak żeby automatycznie przyjmowała kolejne rosnące wartości (PostgreSQL: `SERIAL PRIMARY KEY`, SQL Server: `IDENTITY(1,1) PRIMARY KEY`). Pokazać jak za pomocą języka SQL wstawiać rekordy do tabeli.

   Tabelę napełnić przykładowymi danymi i pokazać podstawowe zapytania typu `SELECT`.

2. (**2p**) Pokazać jak z poziomu kodu node.js pobierać dane z bazy danych (dla PostgreSQL moduł `pg`/`pg-promise`, dla SQL Server moduł `mssql`) korzystając z obiektów typu `Promise` i `async/await`.

   Pokazać jak z poziomu aplikacji node.js wstawić rekord do bazy danych a w aplikacji po wstawieniu pozyskać identyfikator nowo wstawionego rekordu.

   Wskazówka: W przypadku wstawiania rekordu do tabeli z wykorzystaniem sekwencji pozyskanie identyfikatora jest oczywiste, bo i tak najpierw trzeba go pozyskać żeby móc wykonać polecenie `INSERT`. W przypadku autogenerowanej wartości klucza dla PostgreSQL należy skorzystać z klauzuli `RETURNING`, umożliwiającej zwrócenie wartości z zapytania typu `INSERT`, dla SQL Server należy użyć explicite zapytania `SELECT SCOPE_IDENTITY()` po `INSERT`. Szczegóły techniczne doczytać w dokumentacji.

3. (**1p**) Pokazać jak z poziomu kodu node.js wykonać polecenia typu `UPDATE` i `DELETE`.

4. (**3p**) Utworzyć dwie tabele, `OSOBA` i `MIEJSCE_PRACY` połączone relacją wiele-do-jeden (wiele osób może mieć to samo miejsce pracy). Formalnie - w tabeli `OSOBA` dodać kolumnę `ID_MIEJSCE_PRACY`, którą następnie skonfigurować jako **klucz obcy** do tabeli `MIEJSCE_PRACY`.

Pokazać jak z poziomu kodu node.js wygląda scenariusz dodawania nowych rekordów do obu tabel w ramach jednego "procesu biznesowego" - proces powinien najpierw dodać nowe miejsce pracy a następnie identyfikatora nowo dodanego rekordu użyć do dodania nowej osoby.

5. (**3p**) Poprzednie zadanie powtórzyć w scenariuszu w którym tabele `OSOBA` i `MIEJSCE_PRACY` są połączone relacją wiele-do-wiele: osoba może mieć wiele miejsc pracy w tym samym czasie (jak modelować relacje wiele-do-wiele w relacyjnej bazie danych?).

   Podobnie jak poprzednio, zademonstrować kod node.js który w takim złożonym scenariuszu dodaje nowe miejsce pracy i nową osobę ze wskazaniem na miejsce pracy.

6. (**1p**) Zasymulować sytuację w której w tabeli znajduje się dużo rekordów o różnych wartościach (więcej niż powiedzmy milion). Mierzyć czas wykonania prostego zapytania typu `SELECT * FROM OSOBA WHERE Nazwisko='Kowalski'`.

   Następnie zdefiniować indeks na kolumnie użytej w klauzuli wyszukiwania. Powtórzyć pomiar czasu wykonania zapytania.

   Wykonanie zapytania z klauzulą na kolumnie na którą nałożono indeks powinno trwać istotnie **krócej**. Jeśli tak jest - to jest to zgodne z oczekiwaniami. Jeśli nie udaje się zaobserwować tego efektu, to co może być tego przyczyną?

Wiktor Zychla