Organizacja danych w bazie danych

Spis treści

Wstęp

Zacznijmy od tego, że wiemy na tym etapie, że podczas pracy z systemem zarządzania bazami danych, możemy tworzyć wiele baz danych w jednym systemie zarządzania bazami danych. Czyli, gdy korzystamy z PostgreSQL, to możemy stworzyć wiele różnych instancji bazy danych, które będą stworzone na tym samym serwerze bazodanowym. Dotychczas tworzyliśmy bazy danych, które nosiły nazwy takie jak np: **zajavka** lub **zajavka_store**. Baza danych była tworzona przy wykorzystaniu komendy:

```
CREATE DATABASE zajavka;
```

W poniższej notatce poruszymy tematy dotyczące organizacji naszych danych, a mianowicie będą to **schematy bazodanowe** (*database schema*), **indeksy** (*database index*) oraz **widoki** (*database view*).

Jeżeli kiedyś spotkasz się ze zbiorczym stwierdzeniem *database objects*, będzie to oznaczało zbiór elementów takich jak np.

- Tabele - to już znamy,
- Constraints (ograniczenia) - z tym też już mieliśmy do czynienia,
- Indeksy - o tym powiemy za moment,
- Widoki - o tym również powiemy za moment.

Zacznijmy natomiast od tego, czym jest **schemat** w bazie danych.

Database Schema

W obrębie bazy danych możemy tworzyć schematy. Wcześniej nie tworzyliśmy schematów ręcznie, bo cały czas pracowaliśmy na schemacie **public**.

Schemat w bazie danych jest przestrzenią, która zawiera tabele, funkcje, procedury, widoki (nie mówiliśmy o tym), czy indeksy (o tym również nie mówiliśmy).

Baza danych może zawierać jeden lub więcej schematów, a każdy schemat należy tylko do jednej bazy danych. Dwa schematy mogą wtedy posiadać dwie tabele o tej samej nazwie, w tej samej bazie danych. Jeżeli wtedy będziemy odwoływali się do takiej tabeli, to musimy powiedzieć, w którym schemacie się ona znajduje. Czyli jeżeli tabela `employee` będzie znajdowała się w schemacie `public` to odwołamy się do niej w ten sposób:

Poniższe przykłady będą opierały się o encję `Employee`.

```
SELECT * FROM public.employee;
```

Przy czym, jeżeli odwołujemy się do schematu **public**, to nie musimy podawać tej nazwy, bo jest to schemat domyślny.

Jeżeli natomiast będziemy chcieli odwołać się do takiej tabeli w schemacie `company`, to zrobilibyśmy to w ten sposób:

```
SELECT * FROM company.employee;
```

Schematy pomagają nam lepiej zorganizować tabele w bazie danych, np. w logiczne grupy. Schematy pozwalają również na dostęp do jednej bazy danych wielu użytkownikom w taki sposób, że nie będą oni kolidować ze sobą w pracy z danymi - bo pracują na różnych schematach.

W praktyce spotkasz się z takim wykorzystaniem bazy danych, że kilka aplikacji będzie pracowało z tą samą bazą danych, ale każda aplikacja będzie miała na tej samej bazie danych swój własny schemat. Wtedy aplikacje (które możemy rozumieć jak użytkowników bazy danych) nie będą ze sobą kolidować.

Domyślny schemat w PostgreSQL to `public` i jeżeli pracujemy z tabelami w tym schemacie, to nie musimy podawać nazwy `public`. Z drugiej strony, jeżeli nie będziemy jej podawać, to domyślnie będziemy pracować na schemacie `public`.

Schemat w bazie danych tworzymy w ten sposób:

```
CREATE SCHEMA company;
CREATE SCHEMA offers;
CREATE SCHEMA users;
```

Jeżeli stworzymy wiele schematów w bazie danych, to podczas pracy z taką bazą, musimy podawać

schemat, do którego się odwołujemy. Inaczej będziemy pracowali ze schematem `public`.

Database Index

W skrócie można powiedzieć, że **indeks** w bazie danych służy do przyspieszenia wykonywania zapytań. Dodając indeks, zapewniamy mechanizm, dzięki któremu dane w tabeli mogą być wyszukiwane szybciej. Można powiedzieć jeszcze inaczej, że indeks jest wskaźnikiem do danych w tabeli. Indeks w bazie danych możemy rozumieć analogicznie do indeksu z tyłu książki (skorowidz).

Indeks w bazach danych jest specjalną strukturą, która umożliwia szybki dostęp do określonych informacji bez konieczności odczytania wszystkich danych przechowywanych w określonej tabeli.

Przykład z życia

Wyobraź sobie, potrzebujesz szybko znaleźć jakąś informację. Zapomnij, że masz Internet, telefon, masz przed sobą tylko książkę, w której znajduje się opis zagadnienia, którego szukasz. Z racji, że potrzebujesz znaleźć takie dane możliwie szybko, domyślasz się, że odpada przeczytanie całej książki, żeby znaleźć wyjaśnienie pewnego zagadnienia. W takiej sytuacji powinieneś / powinnaś wykorzystać indeks dostępny z tyłu książki (skorowidz). Taki indeks z tyłu książki zawiera posortowaną alfabetycznie listę zagadnień, a przy każdym zagadnieniu numer strony, na której znajdziesz wyjaśnienie. W ten sposób możesz szybko znaleźć, na której stronie będzie wyjaśnione zagadnienie, które Cię interesuje.

Indeks zawiera zatem wszystkie niezbędne informacje do uzyskania szybkiego i wydajnego dostępu do danych.

Potrzeba

Tabele w bazach danych są bardzo podobne do książek. Dane wstawiane do tabel przechowywane są wewnętrznie w blokach, gdzie taki blok może być rozumiany jako ciągły zestaw bajtów na dysku. Nie ma tutaj jakiejś określonej kolejności. Dane wstawiane są do bloku, aż ten się wypełni. Gdy się wypełni, nowy blok przypisywany jest do tabeli i dane są wstawiane do tego nowego bloku. Gdy kasujemy dane, są one usuwane z takiego bloku i pozostaje tam puste miejsce, do momentu dodania kolejnych danych do tabeli - wtedy takie puste miejsce może być wykorzystane przez nowe dane.

Jeżeli nie stosujemy indeksów i potrzebujemy znaleźć jakąś konkretną informację, musimy przeczytać całą "książkę" od początku do końca. Nawet jeżeli w dzisiejszych czasach mamy dość szybkie komputery, to przeczytanie np. 10 milionów rekordów będzie już zauważalne.

W tym właśnie celu wykorzystywany jest mechanizm indeksów, w ten sposób możemy umożliwić szybki dostęp do danych w tabelach. Indeksy w bazach danych działają analogicznie do indeksów w książkach. Indeks taki przechowuje wartości, które są zapisane w kolumnie wraz z odniesieniem do bloków, w których przechowywane są konkretne wartości w tabeli.

Indeksy są domyślnym mechanizmem silników bazodanowych. Za każdym razem, gdy wykonujemy na tabeli INSERT, UPDATE lub DELETE, te same operacje będą wykonywane na indeksie tabeli.

Minusy

Należy jednak pamiętać, że indeks to dodatkowa struktura danych. Oznacza to, że potrzebuje on dodatkowego miejsca na dysku oraz w pamięci. Nadmiarowe tworzenie niepotrzebnych indeksów

spowoduje większe zużycie miejsca na dysku i większe zużycie pamięci, chociażby z tego powodu, że przy aktualizowaniu danych w tabeli, aktualizowane są również indeksy. Indeks tak samo, jak tabela, również wykorzystuje bloki do przechowywania informacji, zatem niepotrzebne definiowanie indeksów zużywa miejsce na dane, które mogłyby być umieszczone w tabelach.

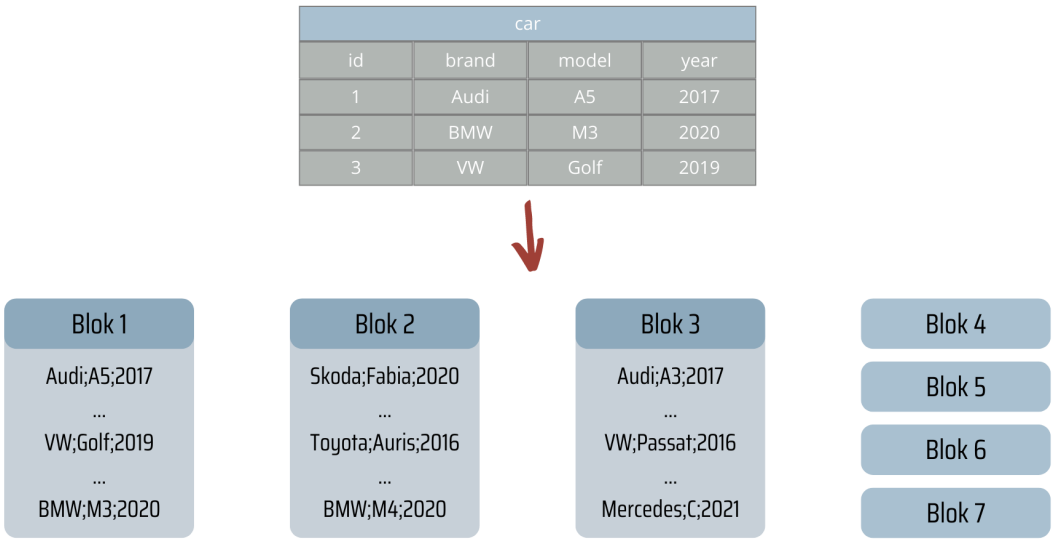
Drugi aspekt jest taki, że skoro indeksy są automatycznie zarządzane przez silnik baz danych, oznacza to, że przy każdej operacji UPDATE, INSERT lub DELETE, silnik taki musi wykonać dodatkowe operacje. Czyli jeżeli użytkownik zmieni numer telefonu, to baza danych musi zaktualizować blok, w którym przechowywany był stary numer telefonu i zaktualizować wszystkie indeksy, gdzie trzeba wymienić stary numer telefonu na nowy. Wszystko to wymaga czasu procesora i dostępu do dysku. Dlatego właśnie nie możemy zastosować indeksu na każdej możliwej kolumnie w tabeli, bo spowolni nam to znacznie działanie bazy danych.

Nie różni się to od innych technik, które poznajemy, wszystko jest fajne, o ile używamy tego zgodnie z przeznaczeniem i rozumiemy co robimy.

Przykład

Na poniższym przykładzie wyjaśnimy sobie schematycznie, jak taki indeks działa i jak pomaga redukować ilość operacji, które muszą być wykonane, żeby odczytać dane z tabeli.

Dane w tabelach nie są przechowywane w żaden konkretny przewidywalny sposób. Jeżeli oprzemy się o to, że dane są przechowywane w blokach, to schematycznie moglibyśmy sobie to wyobrazić w ten sposób:

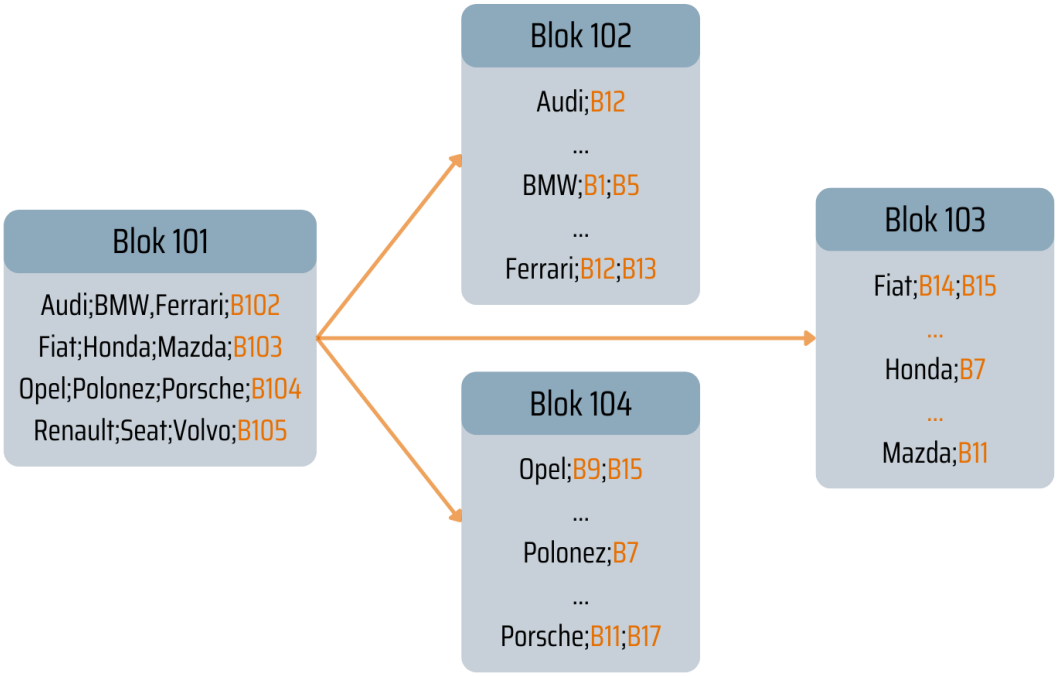


Obraz 1. Tabele i bloki

Załóżmy, że w naszym schemacie, dane w tabeli zostały rozdzielone na dysku na 7 bloków. W praktyce takich bloków mogłoby być znacznie więcej.

Widać zatem na tej podstawie, że jeżeli nie stosowalibyśmy indeksów na tabeli `car`, to szukając np. wszystkich samochodów marki `BMW`, musielibyśmy przeszukać wszystkie rekordy we wszystkich blokach.

Indeksy bazodanowe są implementowane przy wykorzystaniu struktur drzewiastych (przypomnij sobie czym była struktura drzewa w odniesieniu do struktur danych). Gdy tworzymy indeks na podstawie kolumny w tabeli lub kilku kolumn, to taka kolumna lub kolumny są wykorzystywane jako klucz w takim indeksie. Spójrz na poniższą grafikę:



Obraz 2. Schemat działania indeksu

Na grafice powyżej, mamy stworzony indeks na podstawie kolumny `brand`, gdzie blok będący korzeniem drzewa zawiera zakres wartości (tak jak indeks w książce) oraz wskaźniki na konkretne bloki będące liśćmi drzewa. Wartości z podanego zakresu mogą zostać znalezione w liściach drzewa. Liście drzewa zawierają informacje o konkretnych blokach, w których przechowywane są faktyczne dane z tabel w bazie danych.

Jeżeli szukalibyśmy samochodu o konkretnej marce i nie mieli dodanego indeksu na kolumnie `brand` to musielibyśmy przeszukiwać wszystkie dostępne bloki, żeby znaleźć interesujące nas dane. Jeżeli natomiast na kolumnie `brand` dodamy indeks, to poszukiwania będą wyglądały w ten sposób:

- Najpierw odczytaj dane z korzenia drzewa, aby znaleźć, do którego liścia dalej należy się udać w celu znalezienia konkretnej wartości,
- Następnie odczytaj zawartość liścia, żeby znaleźć wskazania na konkretne bloki, w których znajdują się dane,
- Odczytaj dane z końcowego bloku z danymi.

Jeżeli tabela jest duża, to może się okazać, że indeks wymaga węzłów, czyli takich kroków pośrednich pomiędzy korzeniem a liśćmi, w celu przechowywania większej ilości wartości.

Składnia

W zależności od wykorzystywanego silnika baz danych składnia wyrażenia do tworzenia indeksów może być różna, ale w przypadku PostgreSQL wygląda ona w ten sposób:

```
CREATE INDEX idx_car_brand ON car(brand);
CREATE INDEX idx_car_brand_model ON car(brand, model);
```

Indeksy mogą być tworzone na więcej niż jednej kolumnie. Trzeba jednak pamiętać tutaj o pewnej zależności. Jeżeli zdefiniujemy indeks na dwóch kolumnach, np. `brand` oraz `model`, a zapytanie będzie

filtrowało dane tylko po jednej z nich, np. `brand`, to indeks nie zostanie wykorzystany. Jeżeli chcemy, żeby taki indeks został wykorzystany, musimy napisać zapytanie uwzględniające obie kolumny.

Dlatego właśnie warto dobierać tworzone indeksy z rozsądkiem i dobrze się zastanowić, na których kolumnach taki indeks powinien być dodawany.

Kiedy warto jest tworzyć indeksy

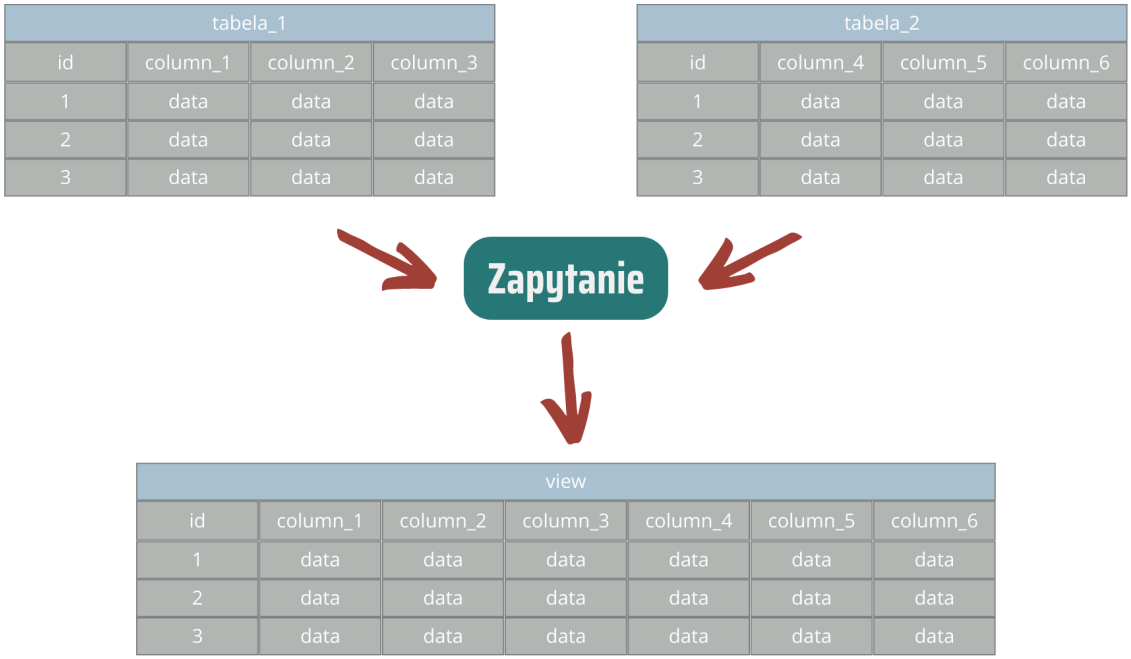
Pamiętaj, że baza danych tworzy indeksy automatycznie dla kluczy głównych. Jest to w sumie logicznie, bo można powiedzieć, że *pkey* jest pierwszym parametrem, po którym szukalibyśmy wartości w bazie danych. Jeżeli natomiast częściej wyszukujemy dane w tabelach na podstawie np. maila użytkownika, warto jest dodać indeks na kolumnie `email`.

Drugim miejscem, gdzie warto jest rozważyć stosowanie indeksów są klucze obce. Indeksowanie kluczy obcych może znacznie poprawić wydajność zapytań, które obejmują JOINy między tabelami. Dodanie takiego indeksu może zapobiec sytuacji, w której tabela w relacji będzie przeszukiwana w całości, żeby znaleźć poszukiwaną wartość. I tutaj właśnie mamy różnicę pomiędzy *pkey* a *fkey*. Bazy danych tworzą automatycznie indeksy dla kluczy głównych, ale dla kluczy obcych już nie.

Oczywiście jak z każdą omawianą techniką, trzeba jej używać z głową. Jeżeli w tabeli będzie umieszczone tylko 10 rekordów, to tworzenie takiego indeksu może się okazać przesadzone, ale gdy w grę będą wchodziły już setki tysięcy rekordów, to wykonanie takiej drobnej komendy może przełożyć się na potężny wzrost wydajności aplikacji. Wszystko to jest kwestia kontekstu.

Database View

Widok w bazie danych może być rozumiany jak obiekt dający możliwość wyszukiwania, który jest zdefiniowany przez określone zapytanie. Widok nie jest tabelą, nie przechowuje danych, jest pewnego rodzaju oknem na dane. Czasami można spotkać się ze stwierdzeniem mówiącym, że: *widok to wirtualna tabela*. Widok może łączyć dane z kilku tabel, a może również ograniczać widoczność pewnych danych, dając tylko ich podzbiór.



Obraz 3. Widok bazodanowy

Widok jest takim obiektem, który możemy utworzyć przy wykorzystaniu jakiegoś zapytania. Często widoki tworzone są na podstawie bardzo skomplikowanych zapytań. Widok może być utworzony na podstawie jednej lub więcej tabel. Z widoku możemy pobierać dane przy wykorzystaniu zapytań *SELECT*. Widoki są tylko do odczytu oraz nie zajmują dodatkowego miejsca na dysku.

Komenda

Jeżeli chcemy utworzyć widok w bazie danych, wykorzystujemy do tego komendę:

```
CREATE VIEW employee_view AS
SELECT id, email
FROM employee
WHERE salary > 1000;
```

Po wykonaniu powyższej komendy, w bazie danych zostanie utworzony widok zawierający dane określone poprzednim zapytaniem. Możemy teraz z takiego widoku pobrać dane:

```
SELECT email
FROM employee_view
ORDER BY email DESC
```

Pobrane dane będą ograniczone tylko do danych przygotowanych przez zapytanie konstruujące widok.

Zalety stosowania widoków

- Jeżeli chcemy "zapisać w bazie danych" jakieś skomplikowane zapytanie, żeby mieć pod ręką dostęp do jego rezultatu, jest to dobry mechanizm, żeby taki efekt osiągnąć. Dane są przechowywane w tabeli źródłowej, zatem nie powielamy ich na potrzeby widoku, a możemy mieć dostęp do specjalnie

przygotowanych danych szybciej i wygodniej,

- Widoki zajmują bardzo mało miejsca, bo dane są przechowywane w tabeli źródłowej,
- Możemy w ten sposób ograniczyć dostęp do danych wrażliwych. Wykorzystując mechanizmy uprawnień użytkowników w bazach danych, możemy ograniczyć w ten sposób dostęp do danych poufnych, wystawiając tylko dane "publiczne" w widokach,
- Możemy uprościć w ten sposób logikę skomplikowanych zapytań, bo będzie ona zapisana w widoku i możemy do tak przygotowanych danych odwoływać się w dowolnym momencie,

Wady stosowania widoków

- Widok nie przechowuje danych. Jeżeli definicja widoku będzie zawierała bardzo skomplikowane zapytanie, może to być dużym obciążeniem dla bazy danych. Wynika to z tego, że za każdym razem, gdy odpytujemy widok, wywoływane jest zapytanie będące jego definicją.
- Widoki są tylko do odczytu, więc teoretycznie możemy potraktować to jako wadę, że nie możemy do widoku wstawiać danych - nie jest to tabela.