Notatki - Database intro

Spis treści

Czym są bazy danych? Co to i po co to?

Strzelając na początku definicją z Wikipedii (angielskiej):

> In computing, a database is an organized collection of data stored and accessed electronically from a computer system.

Czyli baza danych jest takim tworem/programem/systemem, który pozwala nam przechowywać dane. Możemy odczytywać te dane, zapisywać, edytować oraz usuwać.

Dzięki tym operacjom powstał taki skrót w programowaniu jak **CRUD** - **create**, **read**, **update**, **delete**.

Dane w bazach danych są w pewien sposób zorganizowane, mogą być podzielone na kolumny i wiersze, wiersze mogą mieć swoje indeksy, wszystko to po to aby łatwiej móc takimi danymi zarządzać. Oczywiście nie jest to jedyny możliwy sposób organizacji danych, ale na początek będziemy rozmawiać o organizacji w wiersze i kolumny.

DBMS

Skrót **DBMS** rozwija się jako `Database Management System` - czyli oprogramowanie, które służy jako interface pomiędzy użytkownikiem a bazą danych. Inaczej mówiąc jest to oprogramowanie, które umożliwia użytkownikom definiowanie, tworzenie, utrzymywanie i kontrolowanie dostępu do bazy danych. Nazwa takiego systemu świadczy też o tym o jakim serwerze bazy danych rozmawiamy. System zarządzania baz danych pozwala nam tworzyć konkretne bazy danych oraz na nich operować.

W dalszej części wypisane zostały różne systemy zarządzania bazami danych.

Database server

Jeżeli chcemy być bardzo dokładni to wyróżnimy też stwierdzenie serwera bazy danych. Zanim natomiast przejdziemy do wyjaśnienia tego terminu, powiedzmy sobie czym w ogóle jest serwer. A jego definicja jest taka (Oczywiście angielska Wikipedia 😀).

> In computing, a server is a piece of computer hardware or software (computer program) that provides functionality for other programs or devices, called "clients".

Czyli skoro serwer służy do pełnienia jakiejś funkcji dla innych programów lub urządzeń, to serwer bazodanowy jest takim serwerem, który dostarcza nam funkcjonalność bazy danych. Dlaczego tak mieszam?

Nazwa konkretnego **DBMS** świadczy też o tym o jakim serwerze bazy danych rozmawiamy - pomimo, że teoretycznie **DBMS** i serwer bazy danych to są inne pojęcia. Ale jak w praktyce usłyszysz nazwy takie jak np. **PostgreSQL**, **MySQL** lub **Oracle Database** to będą one często oznaczały serwer baz danych i system zarządzania bazami danych.

A czym są te dane?

Imiona, nazwiska, produkty zakupione w internecie, ceny produktów, historie zakupów, daty zakupów, ilość zakupionych produktów i tak dalej... To wszystko to są dane. Jeżeli zapisywalibyśmy takie dane na kartkach, w pewnym momencie zacząłby być potrzebny jakiś system organizacji takich informacji, bo chyba ciężko by było zapisywać te dane na kartkach, a następnie przeszukiwać przykładowo 800 kartek w poszukiwaniu jakiejś informacji.

Naturalnie wypłynęłaby wtedy kwestia takiej organizacji kartek, żeby móc pewne informacje szybko znaleźć. Podzielilibyśmy te kartki na grupy, trzymali w koszulkach, w segregatorach, koszulki byłyby oznaczone karteczkami samoprzylepnymi, moglibyśmy posortować grupy alfabetycznie i tak dalej.

Relacyjne bazy danych

Na potrzeby tego warsztatu skupimy się tylko na relacyjnych bazach danych (ze względu na to jak często są spotykane w praktyce). Relacyjna baza danych (**Relational database**) prezentuje dane w strukturze tabel. Każda tabela składa się z wierszy oraz kolumn. Możesz to sobie wyobrazić w formie pliku w excelu. Możemy w nim stworzyć tabelki, które będą miały określoną ilość kolumn, do tego będziemy wypełniać te tabelki kolejnymi wierszami, w których będą znajdowały się nasze dane.

Oprócz baz relacyjnych występują też nierelacyjne bazy danych, które przechowują dane w innych strukturach niż tabelki, natomiast na ten moment nie poruszamy tego tematu. Takie rodzaje baz danych często nazywane są NoSQL.

A co z tym JDBC?

Java Database Connectivity - bo takie jest rozwinięcie tego skrótu. JDBC to API, które zapewnia nam podstawowe klasy i metody, pozwalające na podłączenie się do bazy danych przy wykorzystaniu Javy.

Jeszcze jakieś ważne skróty?

SQL - **Structured Query Language** - język zapytań pozwalający na zapytanie bazy o dane, które nas interesują. Pozwala również na dodawanie, modyfikowanie oraz usuwanie danych. Będziemy go używać w kolejnych przykładach.

JPA - **Java Persistence API** - zanim wyjaśnię czym jest JPA, to zaczniemy od czegoś innego. W praktyce bardzo często nie używa się JDBC, tylko warstwę abstrakcji wyżej. Na czym polega ta warstwa wyżej? W skrócie chodzi o to, że JDBC jest mechanizmem, który pozwala nam komunikować się z bazami danych. Natomiast dużo rzeczy musi wtedy być robione ręcznie (zobaczysz niedługo). Aby to ułatwić/obejść/przyspieszyć powstało coś takiego jak JPA, o którym będziemy rozmawiać później. Natomiast JDBC trzeba poruszyć, żeby zrozumieć w jaki sposób Java komunikuje się z bazami danych. Pierwszym przykładem, który będzie dosyć denerwujący jest to jak dane z tabelki przemapować na obiekty.

RDBMS - **Relational Database Management System** - do poprzedniego skrótu DBMS dołożyliśmy jeszcze kwestię relacyjności wynikającej z koncepcji relacyjnych baz danych i w ten sposób powstał ten skrót.

Popularne DBMS

Na rynku dostępnych jest dużo popularnych DMBS. Jak zaczniesz szukać w Internecie to często będą pojawiały się takie nazwy jak PostgreSQL, MySQL, Oracle Database lub MariaDB. Podałem nazwy 4 różnych produktów, ale nie przejmuj się, nie musisz znać wszystkich od podszewki. Założenie jest takie, że bazy SQL mają wspierać język SQL, a my jako programiści powinniśmy umieć się SQLem posługiwać, żeby móc z tymi bazami gadać. Jeżeli potrafimy operować językiem zapytań baz danych (SQL) to z każdą z nich powinniśmy być w stanie porozmawiać ☺. Jednakże trzeba pamiętać, że w niektórych przypadkach, składnia tego języka w zależności od tego z której bazy danych korzystamy może się nieznacznie różnić, ale to zaczniesz zauważać już w praktyce.

Jak przechowywane są dane

W tabelach. Wydaje mi się, że ten sposób dla człowieka jest dosyć naturalny. W przypadku baz danych, o których będziemy tutaj rozmawiać, dane są przechowywane w formie tabelarycznej. Przykładowo, jak może wyglądać tabela USERS reprezentująca użytkowników jakiegoś systemu:

Id	Name	Surname	City
1	Stefan	Romański	Warszawa
2	Roman	Stefański	Szczecin
3	Aleksandra	Iksińska	Zakopane
4	Stefania	Stefańska	Wrocław
5	Aleksander	Romanowski	Poznań
6	Anna	Zajavkowa	Gdynia

Kolumna

Kolumna zawiera w sobie informacje określonego typu i jej przeznaczeniem jest przetrzymywanie tylko danego typu informacji. Oznacza to, że w kolumnie `City` nie będziemy wpisywać nazwisk i odwrotnie.

Wiersz

Wiersz jest pojedynczym wpisem w tabeli. W tabeli powyżej mamy 6 wierszy, gdzie każdy z nich reprezentuje inny wpis. Oznacza to, że każdy wiersz reprezentuje inną osobę. Wiersz może być też określany słowem **wpis**. W angielskim możemy spotkać takie słowo jak **record** albo **entry**. Tak jak w Mapach, pamiętasz `Map.Entry`?

Wartości w wierszach mogą być NULL

Tak samo jak referencja może nie wskazywać na żaden obiekt i wtedy mamy w Javie zapis:

```
String value = null;
```

Tak samo w przypadku danych w bazach danych, informacja w danej kolumnie może być `NULL`, czyli brak informacji. Tak jak w przypadku Javy, należy pamiętać, że `null` nie oznacza pustego Stringa tylko brak informacji - tak samo jest tutaj. Jeżeli w wierszu, w kolumnie numerycznej (bo kolumny mają określone typy danych, do których przejdziemy) określimy, że spodziewamy się danych numerycznych, to `null` oznacza brak danych, a jakąś wartość domyślną, np. `0`.

Integralność danych

Możemy poszukać definicji w Internetach "What is data integrity" i cytując angielską Wikipedię:

> Data integrity is the maintenance of, and the assurance of, data accuracy and consistency over its entire life-cycle and is a critical aspect to the design, implementation, and usage of any system that stores, processes, or retrieves data.

Co to oznacza w praktyce? Powstały pewne zasady, które znajdziesz poniżej, które pomagają nam utrzymać porządek w naszych danych. Przestrzegając tych zasad dążymy do tego, żeby nasze dane były kompletne i spójne, co jest krytycznym aspektem w przypadku przechowywania jakichkolwiek danych.

Bo wyobraźmy sobie, że organizujemy sobie dane wypisując je na karteczkach w segregatorach. Przykładowo nie miałoby żadnego sensu posiadanie 5 identycznych karteczek z identyczną zapisaną informacją. Albo drugi przykład, jeżeli karteczka `A` mówiłaby, że jakaś informacja jest zapisana na karteczce `B`, a karteczka `B` wylądowałaby w koszu, to nasze dane są niekompletne. Trzymając się zasad wypisanych poniżej dążymy do zachowania integralności naszych danych.

Integralność encji (Entity integrity)

W jednej tabeli nie możemy mieć dwóch identycznych wierszy. Czyli, nie możemy mieć takiej sytuacji:

Id	Name	Surname	City
1	Stefan	Romański	Warszawa
1	Stefan	Romański	Warszawa

Ale taką już jak najbardziej:

Id	Name	Surname	City
1	Stefan	Romański	Warszawa
2	Stefan	Romański	Warszawa

Zwróć uwagę, że w drugim przypadku są to już inne wiersze, bo w kolumnie `Id` wartości są różne.

Integralność domeny (Domain integrity)

Kolumny mają określone typy, tak samo jak w Javie, żeby `Integer` nie przetrzymywał wartości `String` itp. Dzięki temu narzucamy jaki typ danych może być przechowywany w danej kolumnie.

Więzy integralności (Referential integrity)

Tabele mogą mieć wzajemne relacje (dlatego nazywa się to bazami relacyjnymi), to znaczy, że wartości w kolumnie mogą odnosić się do wartości w innych tabelach i być z nimi powiązane. Jeszcze sobie o tym później powiemy. Natomiast kwestia polega na tym, że przykładowo nie możemy usunąć z tabeli wiersza, jeżeli wartości w tym wierszu odnoszą się do wpisów w innych tabelach, które nie zostały usunięte.

Integralność narzucona przez użytkownika (User-defined integrity)

Tutaj dodam, że chodzi o reguły narzucane przez osobę konfigurującą tabele w bazach danych, a nie o użytkownika końcowego, czyli np. klienta banku. Oprócz tego, że możemy powiedzieć, że dane mają być typu tekstowego, to możemy narzucić wymóg, że w danej kolumnie możemy zapisać tylko wyrazy o długości 10 znaków.

Constraints

Czyli przymus, ograniczenie? Nie wiem jak to przetłumaczyć sensownie na polski, bo w praktyce mówi się o konstrejntach (chyba dobrze napisałem ☺).

Definiując kolumny w bazie danych mamy możliwość narzucenia pewnych reguł, które muszą zostać spełnione przez dane, które chcemy w danej tabeli zapisać, aby w ogóle móc je zapisać.

Constrainty mogą być zarówno na poziomie kolumny jak i całej tabeli. Poniżej wypiszemy sobie najczęściej używane:

- **NOT NULL** - constraint mówiący, że w danej kolumnie nie możemy zapisać `NULL`.
- **UNIQUE** - constraint mówiący, że w danej kolumnie dane nie mogą w żaden sposób się powtarzać, muszą być unikalne. Jeżeli dodalibyśmy `UNIQUE` constraint na kolumnie z imionami (patrz tabele wyżej), to nie moglibyśmy w tej samej tabeli zapisać dwóch `Zbyszków`, bo moglibyśmy mieć w kolumnie z imionami tylko jedno imię `Zbyszek`.
- **DEFAULT** - constraint narzucający wartość domyślną, jeżeli sami takiej nie podaliśmy. Czyli jeżeli podczas zapisywania danych w bazie, w konkretnej kolumnie, w której ustawiono wartość `DEFAULT` nie określimy wartości to zapisana zostanie ta `DEFAULT`.
- **CHECK** - constraint sprawdzający czy dane, które chcemy zapisać w danej kolumnie są zgodne z określonym przez nas warunkiem. Czyli przykładowo, czy zapisany tekst jest w formacie `JSON` (wiem, że jeszcze o tym nie rozmawialiśmy, ale podaję to tylko jako przykład).
- **PRIMARY KEY** (Klucz główny) - constraint jednoznacznie identyfikujący unikalny rekord/wiersz w bazie danych, coś jak pesel. Będziemy o tym rozmawiać.
- **FOREIGN KEY** (Klucz obcy) - constraint jednoznacznie identyfikujący odniesienie do wpisu w innej tabeli. Będziemy o tym rozmawiać.