Notatki - SQL podstawy - cz.2

Spis treści

SQL i podstawowa składnia

Przejdziemy do omówienia składni języka SQL. Na potrzeby przykładów będziemy używać PostgreSQL. Wspominam o tym z tego powodu, że wspomniałem wcześniej, że konkretne DBMS mogą różnić się pewnymi niuansami i tak samo typy danych jak i składania zapytań, które są obsługiwane przez konkretne DBMS mogą się nieznacznie różnić. Nie przejmuj się tym, w pracy jak mamy taką sytuację to zwyczajnie trzeba googlować ☺.

Typy danych

Zacznijmy od tego jakiego rodzaju dane możemy zapisywać w konkretnych kolumnach w tabelach. Od razu też zaznaczę, że poruszone poniżej typy nie są wszystkimi możliwymi.

Boolean

nazwa	rozmiar	zakres	komentarz
boolean	1 byte	false lub true	boolean może przetrzymywać wartości true lub false, w kolumnie może też być zapisany null

Typy numeryczne

nazwa	rozmiar	zakres	komentarz
smallint	2 bajty	-32768 do 32767	integer o małym zakresie danych
integer	4 bajty	-2147483648 do 2147483647	najczęściej używany

nazwa	rozmiar	zakres	komentarz
bigint	8 bajtów	-9223372036854775808 do 9223372036854775807	integer o dużym zakresie danych
real	4 bajty	Dokładność 6 cyfr dziesiętnych	Cytując dokumentację: "The data types real and double precision are inexact, variable-precision numeric types". Czyli nie używać tego typu do pieniędzy, gdyż jest on niedokładny. Pamiętasz double i float z Javy vs BigDecimal?
numeric(p,s)	zmienny	Do 131072 cyfr przed przecinkiem oraz do 16383 cyfr po przecinku	'p' oznacza ilość cyfr, a 's' ilość miejsc po przecinku, czyli numeric(5,2) oznacza 5 cyfr, z czego 2 są po przecinku. Ten typ powinien być używany do kalkulacji pieniężnych

Typy tekstowe/znakowe

nazwa	komentarz
char(n)	Ciąg znaków o długości określonej jako 'n'. Jeżeli wstawimy ciąg znaków, który jest krótszy niż 'n', reszta zawartości zostanie dopełniona spacjami. W przypadku próby zapisania dłuższego tekstu dostaniemy błąd
varchar(n)	Ciąg znaków o długości maksymalnej określonej jako 'n'. Jeżeli ciąg znaków jest krótszy niż 'n', nie zostanie on dopełniony spacjami. W przypadku próby zapisania dłuższego tekstu dostaniemy błąd
text	Tekst o zmiennej długości, teoretycznie może mieć nieograniczoną długość

Daty i czasy

nazwa	komentarz
DATE	przechowuje tylko datę
TIME	przechowuje tylko czas
TIMESTAMP	przechowuje oba datę i czas
TIMESTAMP WITH TIME ZONE	przechowuje oba datę i czas z uwzględnieniem strefy czasowej

Tworzenie tabeli

Jeżeli stworzyliśmy już swoją bazę danych, możemy teraz stworzyć w niej tabelki. Zanim natomiast zaczniemy tworzyć tabelki musimy wiedzieć, co tak na prawdę chcemy stworzyć ☺. Załóżmy, że będziemy tworzyć aplikację, w której będziemy przetrzymywać informacje o wypłatach naszych pracowników razem z datą zatrudnienia pracownika. Potrzebna nam zatem będzie tabela w stylu:

ID	NAME	SURNAME	AGE	SALARY	DATE_OF_EMPLOYMENT
1	Aleksander	Wypłata	33	8791.12	2018-03-12
2	Roman	Pomidorowy	43	7612.12	2012-01-01
3	Anna	Rosół	38	5728.90	2015-07-18
4	Urszula	Nowak	39	3817.21	2014-12-15
5	Stefan	Romański	38	9201.23	2020-07-14
6	Jolanta	Kowalska	27	6521.22	2012-06-04

Zanim w ogóle będziemy mogli dodać dane do takiej tabeli to musimy ją stworzyć i określić jaką taka tabela może mieć strukturę. Pamiętasz, że powinniśmy określić jakiego typu dane mogą się znaleźć w każdej kolumnie? Albo, że możemy narzucić aby w danej kolumnie były tylko unikalne wartości?

Aby stworzyć strukturę tabeli musimy wykorzystać instrukcje **DDL** (**Data Definition Language**). W przykładzie, którego potrzebujemy, instrukcja taka może wyglądać w taki sposób:

```
CREATE TABLE EMPLOYEES(
    ID                  INT           NOT NULL,
    NAME                VARCHAR(20)   NOT NULL,
    SURNAME             VARCHAR(20)   NOT NULL,
    AGE                 INT,
    SALARY              NUMERIC(7, 2) NOT NULL,
    DATE_OF_EMPLOYMENT DATE,
    PRIMARY KEY (ID)
);
```

Pojawiło się stwierdzenie `PRIMARY KEY (ID)`. W tłumaczeniu na nasz jest to **klucz główny**, który jest 'zakładany' na kolumnie `ID`. Oznacza to, że wartość w tej kolumnie ma być unikalna dla każdego zapisywanego w tabeli wiersza. **Primary key** oznacza, że dzięki tej wartości możemy unikalnie identyfikować 2 wiersze, które mogą mieć dokładnie te same zawartości w innych kolumnach.

Teoretycznie są bazy danych, które pozwolą nam stworzyć tabelę bez klucza głównego, ale w praktyce tworzy się go zawsze, aby poprawnie odróżnić od siebie rekordy.

Możemy w tym momencie spróbować zobaczyć definicję nowo utworzonej tabeli, ale zanim jednak to zrobimy, upewnijmy się, że uruchomiliśmy/wskazaliśmy odpowiednią bazę danych:

```
\c zajavka
```

Teraz możemy spróbować odczytać informacje o utworzonym schemacie wykorzystując komendę:

```
\d EMPLOYEES;
```

Lub

```
\d+ EMPLOYEES;
```

Możemy również wykorzystać kwerendę SQL określaną jako `SELECT` (niedługo będziemy dużo tego pisać).

```
SELECT *
FROM information_schema.columns
WHERE table_name = 'EMPLOYEES'
   OR table_name = 'employees';
```

Zaznaczam, że na tym etapie w naszej tabelce nie ma jeszcze żadnych danych, jedynie stworzyliśmy schemat/ramkę, w której możemy te dane zapisać.

Usuwanie tabeli

Skoro tabela została utworzona to możemy ją również usunąć ☺.

```
DROP TABLE EMPLOYEES;
```

Edytowanie struktury tabeli

I skoro możemy tabelę zarówno utworzyć jak i usunąć to możemy również aktualizować jej definicję. Należy jednak pamiętać, że jeżeli w danej tabeli mamy już zapisane dane to trzeba uważać z aktualizacją jej schematu. Analogicznie do tabelki w excelu, jeżeli usuniemy całą kolumnę z danymi to dane z tej kolumny też stracimy.

Aby zmienić strukturę tabeli należy wykorzystać komendę `ALTER TABLE`:

```
ALTER TABLE EMPLOYEES
ALTER COLUMN SURNAME DROP NOT NULL;
```

Kod wyżej zmienia kolumnę `SURNAME` z `NOT NULL` na akceptującą `NULL`.