Bazy danych
przed 4 wykładem

Andrzej Giniewicz

22.03.2024

W tym tygodniu zajmiemy się grupowaniem.

Jeśli interesują nas na przykład średnie zarobki w grupie kobiet i mężczyzn, możemy uruchomić dwa zapytania, podobne do poniższych.

```
SELECT AVG(zarobki) FROM tabela WHERE płeć='K';
SELECT AVG(zarobki) FROM tabela WHERE płeć='M';
```

Niekiedy takie podejście jest wystarczające. Co powinniśmy zrobić, gdy interesuje nas suma dni spędzonych na urlopie każdego roku w ciągu ostatnich 2 lat? Jeśli dane mamy odpowiednio przechowane, możemy na przykład napisać

```
SELECT SUM(na_urlopie) FROM tabela WHERE rok=2019;
SELECT SUM(na_urlopie) FROM tabela WHERE rok=2020;
```

Jeśli jednak interesuje nas okres ostatnich 20 lat, taki sposób staje się niepraktyczny. Po pierwsze wczytujemy dane kilkukrotnie, wybieramy inny ich podzbiór i liczymy sumę na wybranych elementach, po drugie, musimy napisać tyle zapytań, ile mamy grup. Z pomocą w takiej sytuacji przychodzi grupowanie.

Zacznijmy od przykładu z płcią.

```
SELECT AVG(zarobki) FROM tabela WHERE płeć='K';
SELECT AVG(zarobki) FROM tabela WHERE płeć='M';
```

Zobaczmy, co się dzieje przy wykonaniu pierwszego z zapytań. Zaczynamy od jakichś danych. Wypiszemy tylko te dane, które nas interesują. Zaczynamy od wczytania całej tabeli. Dla przykładu wygenerowanych zostało 10 wierszy, po 5 dla kobiet i mężczyzn, korzystając z danych statystycznych zarobków brutto na rok 2020 (mediany to odpowiednio 4500 i 5731 dla kobiet i mężczyzn).

płeć	zarobki
K	3789
K	5184
M	6766
M	5145
K	4275
K	3396
M	5663
M	6921
K	3894
M	4974

Przy wykonywaniu pierwszego z zapytań wybieramy podzbiór spełniający warunek `płeć = 'K'`.

płeć	zarobki
K	3789
K	5184
~~M~~	~~6766~~
~~M~~	~~5145~~
K	4275
K	3396
~~M~~	~~5663~~
~~M~~	~~6921~~
K	3894
~~M~~	~~4974~~

Po wykonaniu filtrowania otrzymujemy tabelę

płeć	zarobki
K	3789
K	5184
K	4275
K	3396
K	3894

Na której liczona jest średnia i zwracana jako jedna liczba w tabeli wynikowej

AVG(zarobki)
4107.6

Następnie niezależnie wykonujemy drugie zapytanie, wybierając dane spełniające warunek `płeć = 'M'`

płeć	zarobki
K	3789
K	5184
M	6766
M	5145
K	4275
K	3396
M	5663
M	6921
K	3894
M	4974

Po wykonaniu filtrowania otrzymujemy tabelę

płeć	zarobki
M	6766
M	5145
M	5663
M	6921
M	4974

Na której liczona jest średnia i zwracana jako jedna liczba w tabeli wynikowej

AVG(zarobki)
5893.8

Alternatywne rozwiązanie wykorzystuje grupowania. W grupowaniu podajemy zmienne, względem których tworzymy grupy. SQL w pierwszej kolejności posortuje tabele względem tych zmiennych.

płeć	zarobki
K	3789
K	5184
K	4275
K	3396
K	3894
M	6766
M	5145
M	5663
M	6921
M	4974

Następnie tabela jest rozcinana tak, aby uzyskać kilka niezależnych bloków, w których zmienna sortowana jest stała.

płeć	zarobki
K	3789
K	5184
K	4275
K	3396
K	3894
M	6766
M	5145
M	5663
M	6921
M	4974

Po czym na każdym bloku wykonywane jest zapytanie z funkcją agregującą `AVG`.

płeć	AVG(zarobki)
K	4107.6
M	5893.8

Aby przekazać do SQL takie zapytanie, użyjemy słowa kluczowego `GROUP BY`.

```
SELECT płeć, AVG(zarobki) FROM tabela GROUP BY płeć;
```

W przykładzie z rokiem użylibyśmy kodu

```
SELECT rok, SUM(na_urlopie) FROM tabela GROUP BY rok;
```

Jeśli chcemy pogrupować po kilku zmiennych, wypisujemy je w `GROUP BY` po przecinkach.

Na liście kolumn wyniku, jeśli używamy grupowania, mogą pojawiać się jedynie funkcje agregujące lub takie, które mają stałą wartość w całej grupie. Oczywiście zmienna, po której grupujemy, ma z definicji stałą wartość w grupie — dlatego w przypadku z grupowaniem po płci pojawiły się dwie zmienne, `płeć` i `AVG(zarobki)`. Nie można użyć zmiennej `zarobki` bez funkcji agregującej, ponieważ każda osoba w grupie ma inne zarobki, zatem nie wiadomo, czyje zarobki pojawiłyby się w wierszach wyniku.

Wróćmy na chwilę do przykładu z urlopem. Gdybyśmy byli zainteresowani tylko tymi wynikami, które mają sumę dni na urlopie poniżej 26, nie moglibyśmy użyć w tym celu instrukcji `WHERE`, ponieważ instrukcja `WHERE` filtruje wyniki przed grupowaniem. Aby móc przefiltrować wyniki po grupowaniu, stosujemy słowo kluczowe `HAVING`. Moglibyśmy zapisać

```
SELECT rok, SUM(na_urlopie) FROM tabela
    GROUP BY rok HAVING SUM(na_urlopie) < 26;
```

Dzięki `HAVING` wyniki grupowania zostaną przefiltrowane i zobaczymy tylko te lata, gdy spełniony będzie odpowiedni warunek. Gdybyśmy chcieli przed podzieleniem na lata wybrać dane tylko jednego pracownika, zrobilibyśmy

```
SELECT rok, SUM(na_urlopie) FROM tabela
    WHERE nr_pracownika=42 GROUP BY rok HAVING SUM(na_urlopie) < 26;
```

Jest to moment, w którym musimy powiedzieć nieco o kolejności wykonywania instrukcji w SELECT. Dla poznanych już słów kluczowych zapytanie odbywa się następująco:

1. wczytanie tabeli przy pomocy FROM,
2. wyrzucenie nieinteresujących wierszy przy pomocy WHERE,
3. podział na grupy przy pomocy GROUP BY,
4. wyrzucenie nieinteresujących grup przy pomocy HAVING,
5. wyliczenie wartości kolumn wypisanych zaraz po SELECT,
6. odfiltrowanie duplikatów przy pomocy DISTINCT,
7. posortowanie wyników przy pomocy ORDER BY,
8. wybranie przypadków przy pomocy LIMIT.

Zwróćmy uwagę na ważną rzecz: po pierwsze HAVING i WHERE mogą być w tym samym zapytaniu. Ponieważ mają podobne znaczenie, czyli oba przyjmują warunek filtrujący, mogą się mylić; po drugie, wartości kolumn są liczone dopiero po sprawdzeniu warunku HAVING, zatem wartości funkcji agregujących nie są wtedy jeszcze wyliczone. Dlatego w standardowym SQL **nie można** napisać

```
SELECT rok, SUM(na_urlopie) AS suma_dni FROM tabela
    GROUP BY rok HAVING suma_dni < 26;
```

Zauważmy, że alias suma_dni będzie policzony krok po HAVING, nie jest tam zatem jeszcze zdefiniowany. MySQL oraz MariaDB posiadają rozszerzenie składni, które pozwala na takie użycie aliasów. Użycie aliasu suma_dni < 26 sugeruje, że jest to proste sprawdzenie i porównanie dwóch liczb. W rzeczywistości najczęściej suma_dni < 26 jest tłumaczone na SUM(na_urlopie) < 26, przez co funkcje agregujące liczone są dwa razy — raz przy sprawdzaniu warunków, drugi raz przy wyliczaniu wartości kolumn. Takie „ukrycie" obliczeń, choć czasem wygodne z punktu widzenia składniowego, może utrudniać analizę wydajności zapytania. Pamiętajmy też, że takie użycie aliasu nie będzie działać w innych silnikach baz danych, o ile też nie implementują takiego rozszerzenia standardowej składni.

Dlaczego podjęto decyzję o tym, żeby HAVING wyliczał się przed SELECT? Jest to przydatne, gdy mamy do policzenia kilka funkcji agregujących i wyliczanie ich trwa długo. Wyrzucenie grup, zanim zaczniemy liczyć wszystkie funkcje agregujące w zapytaniu, może nam zaoszczędzić sporo pracy. Niemniej jednak utrudnia to zapis i nieco wydłuża prostsze zapytania, co nie jest dużym problemem, ponieważ proste zapytania i tak już zwykle wykonują się dość szybko i różnica może być niezauważalna.