Bazy danych
przed 12 wykładem

Andrzej Giniewicz

7.06.2024

Dziś zajmiemy się z zaawansowanymi funkcjonalnościami SQL. Omówimy widoki, operacje na zbiorach, wspólne wyrażenia tablicowe, zapytania ze zwijaniem oraz funkcje okienkowe.

1 Widoki

Widoki to zapytania `SELECT` zapamiętane na później. Możemy o nich myśleć, jak o „funkcjach", które definiujemy i zapisujemy, aby potem ich użyć. Swoją składnią oraz przeznaczeniem są zbliżone do tabel tymczasowych tworzonych za pomocą `SELECT`. Piszemy

```
CREATE VIEW nazwa_widoku AS SELECT ...;
```

Różnica pomiędzy tym zapytaniem a

```
CREATE TEMPORARY TABLE nazwa_tabeli AS SELECT ...;
```

jest taka, że w przypadku widoku zapisywany jest kod zapytania, podczas gdy w przypadku tworzenia tabeli tymczasowej, zapisywane są jego wyniki. Dodatkowo tabele tymczasowe znikają po zakończeniu sesji, podczas gdy widoki pozostają zdefiniowane również po zakończeniu sesji — pod tym kątem zachowują się bardziej jak tabele nietymczasowe.

Podczas wykonywania MariaDB wybiera jeden z dwóch algorytmów, które stosuje w przypadku uruchamiania widoków:

- algorytm MERGE — stara się połączyć kod widoku oraz kod używający widoku w jedno zapytanie i przeprowadzić wspólną optymalizację zapytania,
- algorytm TEMPTABLE — uruchamia kod widoku i zapisuje wyniki w tabeli tymczasowej, następnie w miejscu użycia widoku zastępuje jego nazwę tabelą tymczasową i do niej wykonuje dalsze zapytania.

Algorytm MERGE jest szybszy i MariaDB stara się go użyć, jeśli to możliwe. Wiadomo, że algorytm MERGE nie będzie używany, gdy w zapytaniu zdefiniowanym w widoku znajduje się:

1. wyrażenie `GROUP BY`,
2. wyrażenie `DISTINCT`,
3. wyrażenie `UNION`,
4. funkcje agregujące,
5. podzapytania,
6. inne wyrażenia, które powodują utratę relacji 1 – 1 pomiędzy wierszami wyniku i oryginalnej tabeli.

Aby sprawdzić, czy wybrany został algorytm TEMPTABLE, możemy wyświetlić plan zapytania i sprawdzić, czy widok jest oznaczony jako `DERIVED`.

Zaletą widoków w porównaniu do standardowych tabel jest to, że pozwalają zapisać wirtualne tabele, bez duplikacji danych. Możemy wykonywać na nich zapytania jak na zwykłych tabelach, ale zamiast ich zawartości przechowywany jest kod, zatem zawsze korzystają z aktualnej wersji danych, zapisanych we właściwych tabelach.

Widoki są w SQL od lat 80-tych, należą zatem do powszechnie znanych i stosowanych możliwości. Kolejne sekcje dotyczą nowszych funkcjonalności języka.

2 Przekroje i różnice zbiorów

Suma zbiorów, w sensie wyników dwóch zapytań, zapisana za pomocą `UNION` oraz `UNION ALL`, jest dostępna w niemal każdym silniku baz danych. Oprócz sumy zbiorów standard SQL z 1992 roku definiuje operator przekroju i różnicy zbiorów. Są to operatory `INTERSECT` oraz `EXCEPT`. Używamy ich w podobny sposób do `UNION`:

```
(SELECT ...)
INTERSECT
(SELECT ...);

(SELECT ...)
EXCEPT
(SELECT ...);
```

W pierwszym przypadku w wyniku będą te rekordy, które są w rezultatach wszystkich zapytań składowych jednocześnie, w drugim te, które są w pierwszym wyniku, ale nie ma ich w kolejnych. Podobnie jak przy `UNION` musimy zadbać o to, by wszystkie zapytania składowe zwracały te same kolumny w tej samej kolejności.

Przekroje i różnice zbiorów są dostępne w MariaDB od wersji 10.3 (na serwerze mamy wersję 10.5). Niestety nie są dostępne w MySQL, jedynie w komercyjnej wersji bazy danych Oracle.

3 Wspólne wyrażenia tablicowe

Wspólne wyrażenia tablicowe zdefiniowane są w standardzie SQL z 1999 roku. Mają one dwa dość różne warianty — prosty i rekurencyjny. Prosty wariant jest zbliżony do pojęcia widoku, ale nie tworzy go na stałe, tylko na czas działania jednego zapytania

```
WITH ‘nazwa wyrażenia‘ AS (
  SELECT ...
)

SELECT ... FROM ‘nazwa wyrażenia‘ ...;
```

Zwróćmy uwagę na brak średnika po instrukcji `WITH`. Podobny efekt uzyskalibyśmy za pomocą

```
CREATE VIEW ‘nazwa widoku‘ AS SELECT ...;

SELECT ... FROM ‘nazwa widoku‘ ...;
```

jednakże w przypadku widoku, definicja zostałaby również po zakończeniu zapytania.

Nazw zdefiniowanych za pomocą wspólnych wyrażeń tablicowych możemy używać tak samo jak tabel, mogą występować w kilku miejscach zapytania, a nawet w innych wspólnych wyrażeniach tablicowych

```
WITH ‘nazwa wyrażenia 1‘ AS (
  SELECT ...
),
‘nazwa wyrażenia 2‘ AS (
  SELECT ... FROM ‘nazwa wyrażenia 1‘ ...
)

SELECT ... FROM ‘nazwa wyrażenia 2‘ ...;
```

Wyrażenia takie pozwalają zwykle zachować porządek w kodzie i dodatkowo pozwalają na pewne optymalizacje, dzięki możliwości wielokrotnego użycia podzapytania.

Ciekawsze efekty uzyskamy za pomocą rekurencyjnych wspólnych wyrażeń tablicowych. Aby zobaczyć, na czym polegają, najlepiej przeanalizować kilka przykładów. Załóżmy, że mamy tabelę przechowującą krawędzie grafu. W praktyce może to być na przykład spis połączeń.

z	do
Wrocław	Poznań
Poznań	Gdańsk
Wrocław	Katowice
Katowice	Kraków
Katowice	Warszawa
Białystok	Gdańsk
Białystok	Kraków

Załóżmy, że dane te są przechowywane w tabeli Połączenia(z, do). Chcemy napisać zapytanie, które pozwoli wypisać miejscowości, do których da się tym środkiem transportu dojechać z Wrocławia — niekoniecznie bezpośrednio.

```
WITH RECURSIVE dojade AS (
  SELECT z AS do FROM Połączenia WHERE z='Wrocław'
  UNION
  SELECT Połączenia.do FROM Połączenia JOIN dojade ON dojade.do=Połączenia.z
)

SELECT * FROM dojade;
```

Ponieważ powyższe wspólne wyrażenie tablicowe to jest rekurencyjne (`RECURSIVE`), można użyć go od razu wewnątrz definicji. To prowadzi do ciekawych efektów. Popatrzmy na zapytanie w nawiasie, które definiuje wyrażenie. Stanowi ono sumę dwóch zapytań. Pierwsze zapytanie jest warunkiem początkowym, który wybiera pierwsze wiersze do tabeli wynikowej. Drugie zapytanie łączy te miasta, do których da się dojechać z nowymi połączeniami (warunek `dojade.do=Połączenia.z`) i uzupełnia rezultat o miejsca docelowe (`Połączenia.do`). Po jednym kroku w tabeli pojawiły się nowe rekordy, ale ponieważ definicja jest rekurencyjna, nie poprzestaje na tym, tylko uruchamia się dalej. `UNION` dba o to, aby nie było powtórek. Gdy tabela przestaje się zmieniać, zapytanie się kończy, zwracając prawidłowo miasta, do których da się dojechać z Wrocławia

do
Wrocław
Poznań
Katowice
Gdańsk
Kraków
Warszawa

Zwróćmy uwagę, że na liście nie ma Białegostoku, ponieważ połączenia w bazie dla tego miasta były jedynie wychodzące.

Za pomocą wyrażeń rekurencyjnych możemy zdefiniować na przykład ciąg Fibonacciego. Ze względu na ograniczenie typu danych, wypiszemy tylko wyrazy ciągu do 47 włącznie. Przykład ten deklaruje nazwy zmiennych obok nazwy tabeli `fib`, ponieważ pierwszy wiersz zwraca stałe bez nazw — zatem SQL nie wiedziałby skąd wziąć nazwy kolumn.

```
WITH RECURSIVE fib (n, current, next) AS (
  SELECT 1, 0, 1
  UNION
  SELECT n + 1, next, current + next FROM fib WHERE n < 47
)
SELECT n, current AS ‘Fib(n)‘ FROM fib;
```

Podobnie jak ostatnio zaczynamy od pierwszego wiersza zawierającego `1, 0, 1`, czyli $n = 1$, $F_n = 0$, $F_{n+1} = 1$. Każdy kolejny wiersz wyliczany jest za pomocą reguły, która bierze dotychczasową tabelę, dodaje do kolumny n wartość 1, przesuwa F_{n+1} na pozycję wyrażenia F_n a w miejsce F_{n+1} wstawia $F_n + F_{n+1} = F_{n+2}$. Warunek stopu zadany jest przez wyrażenie `WHERE`. Na końcu wybieramy tylko dwie kolumny z wyniku.

4 Zapytania ze zwijaniem

Jeśli wykonujemy zapytanie z grupowaniem i funkcjami agregującymi, możemy skorzystać z tak zwanego zwijania. Jeśli do wyrażenia `GROUP BY` po zmiennych dopiszemy `WITH ROLLUP`, pojawią się dodatkowe wiersze z podsumowaniami pomiędzy grupami. Wyrażeń ze zwijaniem nie można łączyć z `ORDER BY`. Zacznijmy od zapytania bez zwijania. Załóżmy, że zapytanie

```
SELECT x, y, SUM(z) FROM t GROUP BY x, y;
```

zwraca tabelę

x	y	SUM(z)
1	1	7
1	2	32
1	3	3
2	1	42
2	3	100

Oznacza to, że każda z grup posiada odpowiednią sumę wyliczoną w tabeli. Jeśli do zapytania dodamy zwijanie

```
SELECT x, y, SUM(z) FROM t GROUP BY x, y WITH ROLLUP;
```

zobaczymy wynik

x	y	SUM(z)
1	1	7
1	2	32
1	3	3
1	NULL	42
2	1	42
2	3	100
2	NULL	142
NULL	NULL	184

Dodatkowe wiersze oznaczają sumy pomiędzy kategoriami. Wiersz `1 & NULL` mówi, że suma trzech wcześniejszych grup (tak zwana super-suma) wynosi 42 ($7+32+3$). Podobnie wiersz `2 & NULL` podaje sumę wszystkich wierszy, które mają $x = 2$, czyli $42 + 100 = 142$. Ostatni wiersz podaje sumę całej tabeli ($7+32+3+42+100 = 184$). Zwijanie jest niekiedy przydatne, jeśli chcemy wygenerować szybkie podsumowania dla funkcji agregujących.

5 Funkcje okienkowe

Funkcje okienkowe są najnowszym spośród omawianych elementów SQL, pochodzą ze standardu SQL z 2003 roku. Są one konstrukcją pośrednią pomiędzy zwykłymi funkcjami a funkcjami agregującymi — jak zwykłe funkcje, funkcje okienkowe zwracają po jednej wartości dla każdego wiersza, ale jak funkcje agregujące, wyliczane są na podstawie więcej niż jednego wiersza. Do wyliczania wartości funkcji okienkowych trzeba określić:

1. wyliczaną funkcję,
2. partycję,
3. porządek,
4. okno.

Wiele funkcji agregujących może być użytych jako funkcje okna, na przykład `AVG`, ale istnieje też więcej przydatnych funkcji, na przykład kwantyle lub rangi — są to bardzo przydatne opcje w wielu zastosowaniach związanych z danymi. Pełna lista wspieranych funkcji znajduje się w dokumentacji.

Partycje stanowią odpowiednik grupowania, wydzielają zbiór rekordów, na którym działa funkcja okna. W ramach partycji zadawany jest porządek, ustalający kolejność wierszy w partycji. Jest to element konieczny — w przeciwieństwie do grupowania przy pomocy `GROUP BY`, gdy nie mamy gwarancji, co jest pierwszym, a co ostatnim rekordem, przy funkcjach okienkowych musimy określić kolejność rekordów w partycji.

Znów posłużymy się kilkoma przykładami, aby zobaczyć przydatne wykorzystania funkcji okienkowych.

```
SELECT czas, cena, AVG(cena) OVER (
  ORDER BY czas
```

```
    ROWS BETWEEN 3 PRECEDING AND 3 FOLLOWING
  ) FROM giełda ORDER BY czas;
```

Zapytanie takie zwróci trzy kolumny — czas i cenę z tabeli giełda, posortowane po czasie, oraz dodatkową kolumnę — AVG(cena), która będzie zawierać średnią kroczącą. Dla każdego wiersza zostanie ona wyliczona na podstawie okna zawierającego 3 poprzedzające i 3 następujące wiersze po badanym, czyli maksymalnie 7 wierszy. Na krańcach tabeli, dla pierwszego i ostatniego wiersza, obliczenia zostaną wykonane na mniejszej liczbie rekordów, jeśli nie można wybrać „następnych" i „poprzednich" wierszy.

Słowo kluczowe, które informuje, że tworzymy zapytanie z funkcją okienkową jest `OVER`. Następnie w nawiasie definiujemy kolejność rekordów. Nie określamy partycji, więc całe dane stanowią jedną partycję. `ROWS BETWEEN ... AND ...` jest jedną z możliwych specyfikacji okna. W miejsca kropek może trafić:

1. `UNBOUNDED PRECEDING` (początek tabeli),
2. `n PRECEDING` (*n* wierszy wstecz),
3. `CURRENT ROW` (obecny wiersz, dla którego wyliczana jest wartość),
4. `n FOLLOWING` (*n* wierszy wprzód),
5. `UNBOUNDED FOLLOWING` (koniec tabeli).

Aby na przykład zdefiniować sumę wydatków, od początku inwestycji do danego momentu, możemy na przykład napisać

```
SELECT czas, kwota, SUM(kwota) OVER (
  ORDER BY czas
  ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
) FROM inwestycja ORDER BY czas;
```

Takie zapytanie obok każdego wiersza dopisze dotychczasową sumę skumulowaną.

6 Podsumowanie

Standard SQL cały czas się rozwija. Większość materiałów dydaktycznych dotyczy standardu SQL z 1986 roku. Nowsze możliwości SQL są często oznaczane jako „zaawansowane funkcje SQL". Należy pamiętać, że od momentu pojawienia się czegoś w standardzie do implementacji często mija wiele lat, czasem dekady. Funkcjonalność tu omówiona dotyczy implementacji w bazie MariaDB. Inne dialekty SQL mogą mieć dużo różnic pod kątem tych możliwości! Aby dowiedzieć się więcej, sprawdź oficjalną dokumentację:

- widoki (SQL:86) — `https://mariadb.com/kb/en/creating-using-views/`,
- przekroje tabel (SQL:92) — `https://mariadb.com/kb/en/intersect/`,

- różnice tabel (SQL:92) — https://mariadb.com/kb/en/except/,
- wspólne wyrażenia tablicowe (SQL:99) — https://mariadb.com/kb/en/non-recursive-common-table-expressions-overview/,
- rekurencyjne wspólne wyrażenia tablicowe (SQL:99) — https://mariadb.com/kb/en/recursive-common-table-expressions-overview/,
- zapytania ze zwijaniem (SQL:99) — https://mariadb.com/kb/en/select-with-rollup/,
- funkcje okienkowe (SQL:2003) — https://mariadb.com/kb/en/window-functions/.