Bazy danych
przed 9 wykładem

Andrzej Giniewicz

10.05.2024

Dziś nauczymy się mierzyć czas wykonania zapytania i analizować, co SQL wykonuje i jakie decyzje podjął w procesie optymalizacji. Dodatkowo nauczymy się mierzyć czas zapytań.

1 Plan wykonania zapytania

W MariaDB istnieją trzy sposoby na sprawdzenie planu wykonania zapytania. Przed komendą `SELECT` możemy dopisać `EXPLAIN`, `EXPLAIN EXTENDED` lub `ANALYZE`, na przykład

```
EXPLAIN SELECT * FROM sakila.actor WHERE first_name LIKE 'A%';
```

Po wykonaniu takiego zapytania, naszym oczom ukaże się nasz pierwszy plan zapytania. Jest to podsumowanie wszystkich informacji i decyzji, jakie serwer SQL podjął przed wykonaniem zapytania. Ważne, że gdy sprawdzamy plan zapytania, dodając `EXPLAIN` przed zapytaniem, zapytanie nie jest wykonywane. Oznacza to, że możemy analizować plan wykonania, nawet jeśli zapytanie trwa zbyt długo, aby je uruchomić.

Zapytanie z `EXPLAIN` zwraca wiele wierszy, po jednym na zapytanie i tabelę. Kolumny w wyniku są następujące.

id jest numerem podzapytania w zapytaniu. Zwykle kolejne wywołania `SELECT` otrzymują kolejne numery, w kolejności występowania. Jeśli mamy tylko jedno zapytania, numer `id` będzie zawsze równy 1. Jeśli `id` jest równe `NULL`, oznacza to, że mowa o zapytaniu `SELECT`, które nigdzie nie jest napisane wprost. Dzieje się tak przy `UNION`, które tworzy tabelę tymczasową, wykonuje dwa zapytania dodające do niej wiersze, po czym wykonuje kolejne zapytanie odczytujące wiersze z tabeli tymczasowej. Ostatniego z tych zapytań nie ma wprost w zapytaniu, więc miałoby `id` równe `NULL`.

select_type opisuje typ zapytania. Ważniejsze wartości to[1]:

SIMPLE oznacza, że zapytanie nie wykorzystuje podzapytań ani `UNION`;

[1] Pełna lista dostępna na `https://mariadb.com/kb/en/explain/#select_type-column`.

PRIMARY oznacza, że zapytanie wykorzystuje podzapytania lub `UNION`, przy czym jest najbardziej zewnętrznym zapytaniem, które zawiera w sobie pozostałe, lub jest pierwszym zapytaniem w `UNION`, które dyktuje jakie będą typy wyników;

DERIVED oznacza podzapytanie występujące we `FROM`;

SUBQUERY oznacza niezależne podzapytanie występujące w innym miejscu niż we `FROM`;

DEPENDENT SUBQUERY oznacza podzapytanie zależne występujące w innym miejscu niż we `FROM`;

UNION oznacza drugą i każdą kolejną tabelę w zapytaniu `UNION`;

UNION RESULT oznacza zapytanie użyte do pobrania wyniku z anonimowej tabeli użytej do scalania wyników;

table oznacza nazwę tabeli lub jej alias. Niekiedy, gdy zapytanie odnosi się do nieistniejących tabel, używane są specjalne nazwy `<derivedN>` lub `<unionA,B,C,...>`, na przykład `<derived3>` lub `<union1,7>`. Pierwszy z przykładów oznacza tabelę tymczasową zwróconą przez zapytanie o `id` równym 3, w drugim przypadku mamy tabelę tymczasową powstałą z połączenia tabel o `id` równych 1 i 7;

type oznacza, w jaki sposób SQL będzie wyszukiwał wierszy w danej tabeli. Ważniejsze wartości to[2]:

ALL oznacza, że wszystkie wiersze tabeli muszą zostać odczytane — to najmniej efektywny wariant, działający w czasie liniowym lub logarytmiczno-liniowym, jeśli konieczne jest sortowanie. **Ten fragment zapytania to dobry kandydat do optymalizacji**;

index oznacza, że wszystkie wiersze tabeli muszą zostać odczytane, ale wykorzystano klucz, aby przyśpieszyć sortowanie — to zapytanie wciąż może być dobrym kandydatem do optymalizacji, ponieważ wciąż działa w czasie liniowym;

range oznacza, że wykorzystano klucz do tego, aby pominąć część tabeli i odczytać tylko jej wycinek. Wycinek wciąż jest czytany wiersz po wierszu, ale można wykorzystać klucz do sortowania wartości;

ref oznacza, że wykorzystano nieunikatowy klucz do znalezienia konkretnego wiersza, potencjalnie z duplikatami;

eq_ref oznacza, że wykorzystano unikatowy klucz do znalezienia konkretnego wiersza;

system oznacza, że tabela ma co najwyżej jeden wiersz, więc możliwe są bardzo silne optymalizacje;

const oznacza, że tabela ma dokładnie jeden wiersz, zatem zostanie wczytana przed wykonaniem zapytania i potraktowana jako wartości stałe;

[2] Pełna lista dostępna na `https://mariadb.com/kb/en/explain/#type-column`

possible_keys zawiera listę kluczy, które zostały rozważone w zapytaniu w celu jego przyśpieszenia, wartość `NULL` oznacza, że nie było kluczy, które można wykorzystać;

key zawiera klucz, który został wybrany do wykonania zapytania, jako potencjalnie dający największą wydajność zapytania, wartość `NULL` oznacza, że zapytanie nie wykorzystuje żadnego klucza. Istnieje możliwość, że wykorzystany zostanie klucz spoza listy kluczy `possible_keys`, wtedy został wybrany z innych powodów niż przyśpieszenie zapytania;

key_len pojawia się, gdy używamy części klucza składającego się z wielu kolumn, zawiera liczbę wykorzystanych bajtów klucza;

ref zawiera wartość `const`, `func` lub nazwę kolumny — są to wartości, które są porównywane z kluczem w zapytaniu;

rows zawiera oszacowanie, ile wierszy znajdzie każde poszukiwanie wartości w danej tabeli;

extra zawiera flagi, które przekazują dodatkowe informacje. Ważniejsze flagi to[3]:

Using index oznacza, że nie ma konieczności dostępu do tabeli — dzieje się tak, gdy wykorzystujemy tylko kolumny z klucza i nie używamy pozostałych kolumn;

Using where oznacza, że po wczytaniu tabeli będzie konieczne przefiltrowane wierszy za pomocą warunku — niekiedy w zapytaniu jest instrukcja `WHERE`, ale dotyczy klucza i nie posiada skomplikowanych wyrażeń, w takiej sytuacji można uniknąć filtrowania wierszy, tylko można od razu pobrać te interesujące nas wiersze — zwykle wtedy zamiast `Using where` pojawi się `range`, `ref` lub `eq_ref` w typie kolumny;

Using temporary oznacza, że do wykonania zapytania musi zostać wykorzystana tabela pomocnicza;

Using filesort oznacza, że konieczne jest sortowanie wyników w oparciu o algorytm sortowania plików — jest tabela jest mała, całość sortowania zostanie zoptymalizowana i wykona się w pamięci, na „od razu wczytanym pliku”;

Distinct oznacza, że na końcu tego zapytania zostaną usunięte duplikaty.

Opisane powyżej możliwości to tylko niewielki fragment tego, co może pojawić się w analizie planu zapytania, ale pokrywa to większość przydatnych zastosowań. Ogólnie, jeśli widzimy `Using filesort` i `ALL` lub `index` bez `Using index`, prawdopodobnie zapytanie warto przemyśleć i zoptymalizować.

Istnieją dwa sposoby, aby uzyskać jeszcze więcej informacji. Pierwszy to

```
EXPLAIN EXTENDED
```

zawierający bogatszą wersję planu zapytania.

[3] Pełna lista dostępna na `https://mariadb.com/kb/en/explain/#extra-column`

```
EXPLAIN EXTENDED SELECT * FROM sakila.actor WHERE first_name LIKE 'A%';
SHOW WARNINGS;
```

Po pierwsze, do standardowej tabeli z wynikami zapytania trafia dodatkowa kolumna.

filtered oznacza pesymistyczne szacowanie tego, ile wierszy tabeli spełnia warunek `WHERE`.

Po drugie, wykorzystując plan zapytania, przepisuje zapytanie na bliższe temu, co w rzeczywistości się wykona. Przepisane zapytanie emitowane jest jako ostrzeżenie, zatem możemy je wyświetlić komendą `SHOW WARNINGS`. Zapytanie to przykładowo nie będzie miało gwiazdek, tylko wypisane wszystkie pozyskane kolumny w konkretnej kolejności. Jeśli gdzieś występuje optymalizacja `const`, czyli zastąpienie podzapytania wartością stałą, to również w jego miejscu pojawi się już konkretne stała. Zastosowanie tej procedury pozwala nam lepiej zobaczyć, jakie decyzje podjął optymalizator.

Kolejnym sposobem na uzyskanie jeszcze pełniejszej informacji jest użycie komendy `ANALYZE`.

```
ANALYZE SELECT * FROM sakila.actor WHERE first_name LIKE 'A%';
```

Z jednej strony zachowanie komendy `ANALYZE` jest podobne do `EXPLAIN`, ponieważ nie emituje ostrzeżenia z przepisanym zapytaniem. Z drugiej strony, jest podobne do wariantu `EXPLAIN EXTENDED`, ponieważ zawiera wiersz `filtered` z szacowaniem liczby wierszy po warunku `WHERE`. Oprócz tego jednak, że **wykonuje zapytanie**, ignoruje jego wynik i zwraca plan zapytania uzupełniony o rzeczywiste warianty `rows` i `filtered` oparte o pomiary, a nie estymację. Rzeczywiste warianty tych kolumn znajdują się w `r_rows` i `r_filtered`. Niestety, ponieważ `ANALYZE` wymaga uruchomienia zapytania, nie możemy użyć go do diagnostyki, dlaczego jakieś zapytanie działa tak długo, że nie może się zakończyć w rozsądnym czasie[4].

2 Uruchomione zapytania i ich plan wykonania

Istnieje możliwość sprawdzenia, jakie zapytania działają. Wykonując instrukcję

```
SHOW FULL PROCESSLIST;
```

możemy zobaczyć listę sesji uruchomionych na serwerze oraz zadań z nimi powiązanych. Mamy tam numer `Id` procesu, nazwę użytkownika i adres, z którego się łączy, bazę danych i komendę. Interesują nas komendy `Query`, czyli działające zapytania. Widzimy tam również czas w sekundach, przez który proces jest w danym stanie. Po kolumnie `Info` możemy sprawdzić, jakie zapytania się obecnie wykonują. Jeśli odnajdziemy numer działającego zapytania, na przykład działającego bardzo długo, możemy sprawdzić jego plan.

[4]Więcej o interpretacji wyników `ANALYZE` można przeczytać w dokumentacji `https://mariadb.com/kb/en/analyze-statement/`.

```
SHOW EXPLAIN FOR 42;
```

Spowoduje wyświetlenie planu zapytania działającego w procesie numer 42. Własne zapytanie możemy też wyłączyć komendą `KILL 42;`, musimy jednak liczyć się z tym, że zakończone zostanie nie tylko zapytanie, ale cały proces obsługujący naszą sesję, a co za tym idzie, zostaniemy rozłączeni z bazą — pozwoli to jednak posprzątać po sobie, jeśli zostawimy dużo wiszących procesów. Procesy wykonujące komendą `Sleep` nie zajmują dużo zasobów, więc nie musimy ich po sobie sprzątać.

3 Profilowanie

Profilowanie jest procesem mierzenia czasu wykonania zapytania. Ponieważ zbieranie danych profilowania spowalnia zapytania, jest ono domyślnie wyłączone. Możemy je włączyć komendą

```
SET profiling=1;
```

i ponownie wyłączyć komendą

```
SET profiling=0;
```

Zmienna ta jest zmienną sesji, więc po zakończeniu sesji przywracana jest wartość domyślna, czyli profilowanie jest wyłączone.

Po włączeniu profilowania czas zapytań będzie mierzony i zapamiętywany. Informację o zmierzonych zapytaniach możemy wyświetlić komendą

```
SHOW PROFILES;
```

W uproszczonej tabeli zobaczymy numer zapytania, zaczynający się od 1 po włączeniu profilowania, czas w sekundach oraz początek zapytania. Jeśli chcemy, znając numer zapytania, możemy wyświetlić dane szczegółowe.

```
SHOW PROFILE FOR QUERY 1;
```

Po wyświetleniu informacji zobaczymy bardzo precyzyjne wyszczególnienie, które operacje zajęły ile czasu. Nie sposób wypisać ich wszystkich, ponieważ możliwości jest ponad sto. Najlepiej zidentyfikować te wiersze, które zajmują najwięcej czasu, po czym sprawdzić w dokumentacji[5]. Jeśli na przykład okaże się, że większość czasu spędzamy w statusie „Copying to group table", sprawdzamy dokumentację i odkrywamy, że

[5]Lista stanów z opisem jest pod adresem `https://mariadb.com/kb/en/general-thread-states/`, znajdziemy tam większość przydatnych, oprócz „Starting" i „Reset for next command".

> Sorting the rows by group and copying to a temporary table, which occurs when a statement has different GROUP BY and ORDER BY criteria.

Oznacza to, że problematycznym elementem naszego zapytania jest to, że konieczna jest tabela pomocnicza, ponieważ `GROUP BY` i `ORDER BY` mają różne kryteria sortowania. Musimy się zatem zastanowić, czy da się uniknąć zmiany kolejności sortowania? A jeśli nie, to czy da się je przyśpieszyć, na przykład tworząc tabelę tymczasową z indeksem? Wiedząc już, jak mierzyć czas, możemy spróbować obu rozwiązań — zmierzyć dwa nowe i porównać z oryginalnym, aby wybrać najlepsze z nich.

Więcej informacji o profilowaniu znajdziesz w rozdziale 3 książki „High Performance MySQL", natomiast o planach zapytań w rozdziale 6 i dodatku D tej samej pozycji.