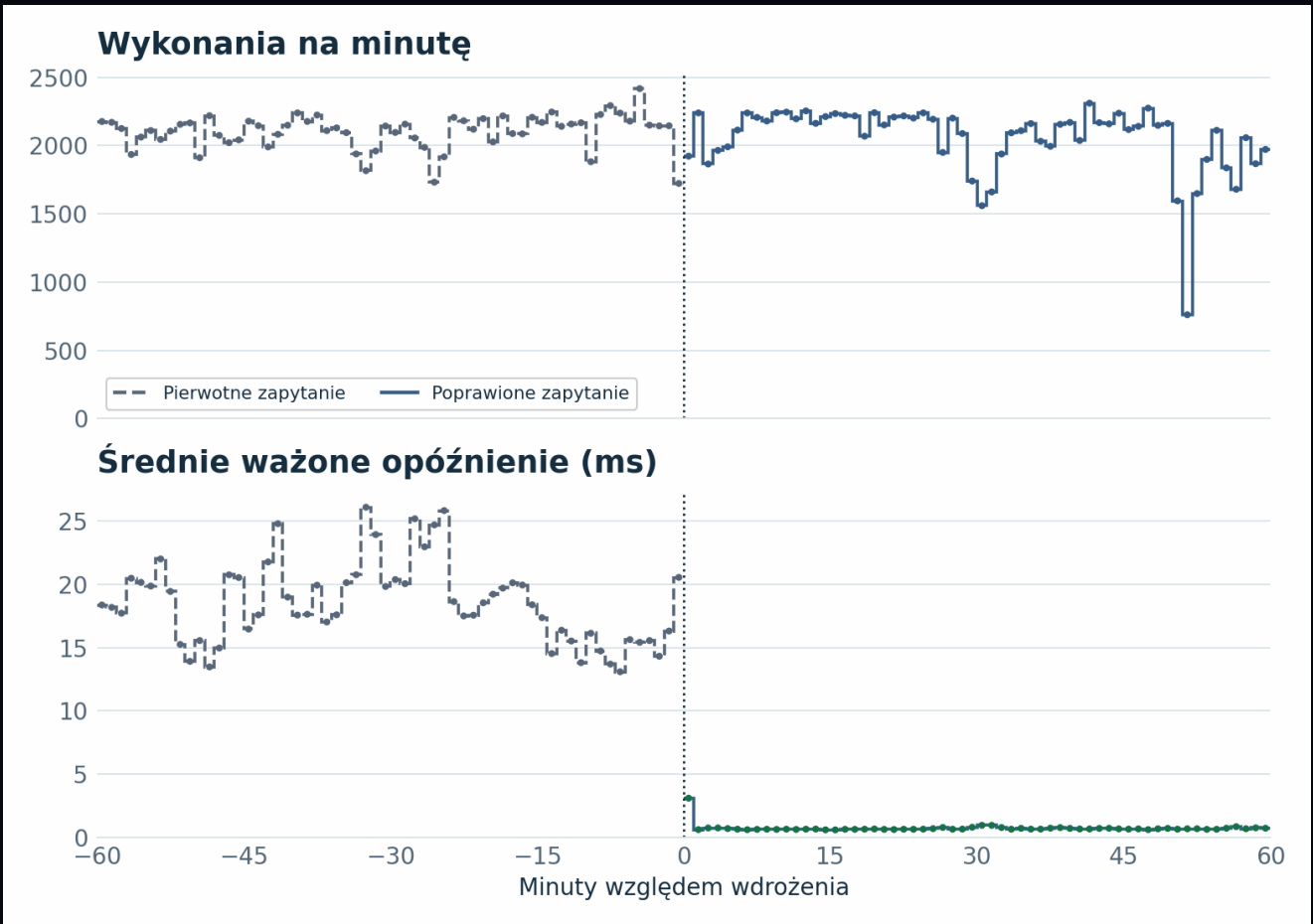
Dlaczego czas zapytania w MariaDB spadł z 18,43 do 0,70 ms, skoro sama nazwa wskazywała już jeden wiersz

Aurélien LEQUOY · November 21, 2026

PMACONTROL MARIADB PERFORMANCE SQL BENCHMARK OBSERVABILITY



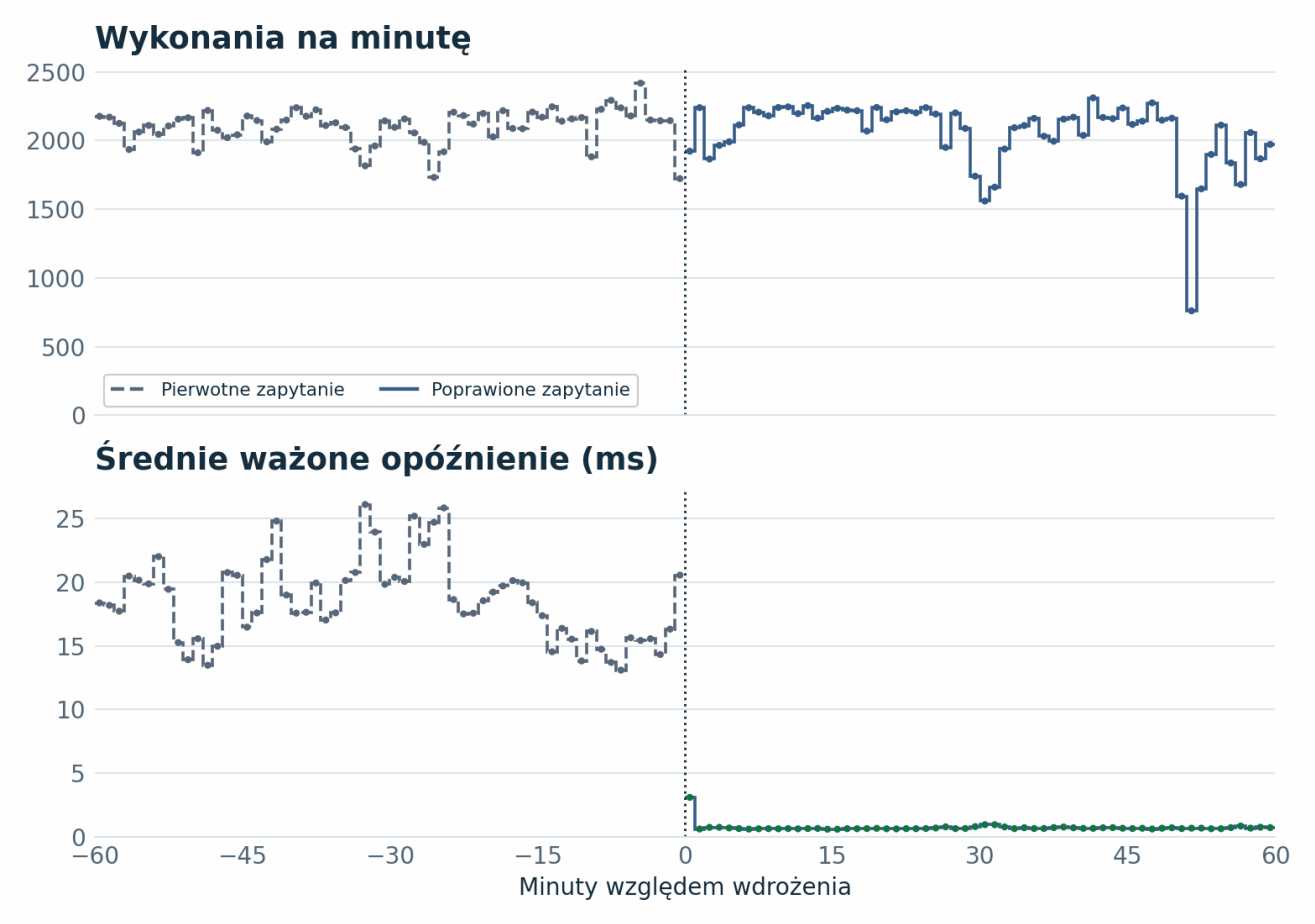
W tabeli `ts_variable` było **3759 różnych nazw na 3791 wierszy**, więc już kolumna `name` zapewniała bardzo wysoką selektywność. Dla `mysql_available` wskazywała dokładnie jeden wiersz. Warunek `v.`from` = 'mysql_server'` wskazał ten sam wiersz. Można się było spodziewać znikomego zysku. Tymczasem przy około 2100 wywołaniach na minutę **średni czas w produkcji spadł z 18,43 do 0,70 ms**. Co zyskała MariaDB, skoro drugi warunek nie wykluczył ani jednego dodatkowego wiersza dla tej metryki?

Przed zapisaniem dostępności usługi PmaControl odczytuje poprzednią wartość, aby wykryć zmianę stanu. W godzinie przed wdrożeniem ten odczyt pochłonął **2326 sekund łącznego**

czasu SQL. W następnej godzinie zużył **87 sekund** przy podobnej liczbie wywołań. Zmienił się jeden warunek złączenia.

Dwie godziny wokół wdrożenia

21 listopada 2026 r. o **17:26:47 czasu paryskiego** kolektor zaczął używać zmienionego zapytania. Nowy tekst SQL otrzymał osobny identyfikator digest w Performance Schema. Na wykresie starego identyfikatora liczba wywołań spada do zera. Dodajemy przyrosty liczników obu wersji w przedziałach jednominutowych, aby śledzić rzeczywiste odczyty usługi.



Każda seria obejmuje 120 punktów, po jednym na minutę. Górny wykres pokazuje liczbę wykonań w każdej minucie, a dolny średnie opóźnienie tych samych wykonań. Pionowa linia oznacza wdrożenie. Uwzględniamy wywołania drugiej wersji w pobliżu przełączenia.

W tabelach zieleń (✓) oznacza poprawę, a czerwień (✗) pogorszenie, gdy większa wartość przekracza dwukrotność mniejszej. Dla czasów różnica musi również wynosić co najmniej 1 ms. Liczba wywołań opisuje obciążenie. Próg ułatwia odczyt wyników. Linie na wykresach rozróżniają okresy. Każdy punkt opóźnienia po wdrożeniu porównano ze średnią ważoną z poprzedniej

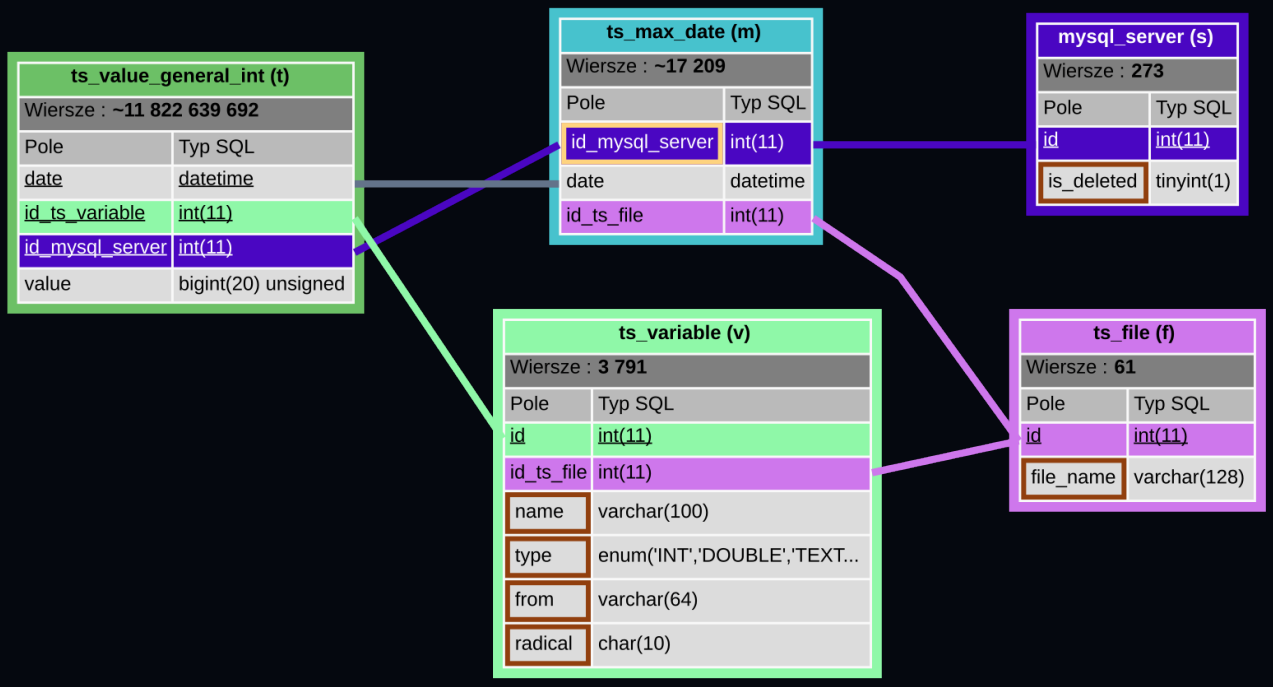
godziny. Wykres łączny odnosi się do tego samego wolumenu, a każda para pomiarów do własnego czasu początkowego.

Miara	Godzina przed	Godzina po
Szacowana liczba wykonań	126 229	123 241
Wykonania na minutę	2104	2054
Średnie ważone opóźnienie	18,43 ms	✓ **0,70 ms (÷26,25)**
Łączny czas SQL w sekundach	2326,5	✓ **86,5 (÷26,88)**
Wykonania na sekundę SQL	54	✓ **1424 (×26,25)**
Badane wiersze na wykonanie	2,99	✓ **1,00 (÷2,98)**

Wydajność wyrażona jako liczba odczytów na sekundę łącznego czasu SQL wzrosła z **54 do 1424** . Opóźnienie spadło o **96,2%**, a obserwowana liczba wykonań na minutę tylko o **2,4%**. Pierwsza miara opisuje koszt odczytu. Rzeczywista przepustowość zależy też od częstotliwości pomiarów i współbieżności.

Jakiej wartości szukamy

Dla wskazanego serwera `ts_max_date` podaje ostatni czas pomiaru danego pliku. `ts_file` identyfikuje plik, a `ts_variable` metrykę. `mysql_server` wyklucza usunięte serwery. Złączenie zewnętrzne pobiera wartość całkowitą dla tego czasu, zmiennej i serwera. Przy brakującym pomiarze może zwrócić `NULL`.



Schemat poprawionego zapytania z PmaControl. Strzałki łączą kolumny używane w złączeniach. Brązowy obrys oznacza pola z warunkami filtrowania, w tym `ts_variable.from`. Znak `~` oznacza liczbę wierszy oszacowaną w chwili odczytu.

Definicje pięciu tabel

Poniższe definicje pochodzą z `SHOW CREATE TABLE` na badanej instancji MariaDB. W definicji `ts_variable` wiersz `UNIQUE KEY` podaje obie kolumny używane przez poprawione zapytanie w kolejności indeksu.

`SHOW CREATE TABLE ts_variable;`

```
CREATE TABLE `ts_variable` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `id_ts_file` int(11) NOT NULL DEFAULT 3,
  `name` varchar(100) NOT NULL,
  `type` enum('INT','DOUBLE','TEXT','JSON','') NOT NULL COMMENT '1 => int, 2 => double, 3 =>
text, 4=> json',
  `from` varchar(64) NOT NULL,
  `radical` char(10) NOT NULL,
  `is_derived` int(11) DEFAULT 1,
  `is_dynamic` int(11) DEFAULT 1 COMMENT 'only for global variables',
  PRIMARY KEY (`id`),
  UNIQUE KEY `name` (`name`,`from`),
  KEY `ts_variable_ibfk_1` (`id_ts_file`),
```

```
  KEY `from` (`from`),
  CONSTRAINT `ts_variable_ibfk_1` FOREIGN KEY (`id_ts_file`) REFERENCES `ts_file` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=39387 DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci;
```

SHOW CREATE TABLE ts_file;

```
CREATE TABLE `ts_file` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `file_name` varchar(128) NOT NULL,
  `each` int(11) NOT NULL DEFAULT 1,
  PRIMARY KEY (`id`),
  UNIQUE KEY `file_name` (`file_name`)
) ENGINE=InnoDB AUTO_INCREMENT=25508473 DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci;
```

SHOW CREATE TABLE ts_max_date;

```
CREATE TABLE `ts_max_date` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `id_mysql_server` int(11) NOT NULL,
  `date` datetime DEFAULT NULL,
  `sample_epoch` bigint(20) unsigned NOT NULL DEFAULT 0,
  `date_p1` datetime DEFAULT NULL,
  `sample_epoch_p1` bigint(20) unsigned NOT NULL DEFAULT 0,
  `date_p2` datetime DEFAULT NULL,
  `sample_epoch_p2` bigint(20) unsigned NOT NULL DEFAULT 0,
  `date_p3` datetime DEFAULT NULL,
  `sample_epoch_p3` bigint(20) unsigned NOT NULL DEFAULT 0,
  `date_p4` datetime DEFAULT NULL,
  `sample_epoch_p4` bigint(20) unsigned NOT NULL DEFAULT 0,
  `id_ts_file` int(11) NOT NULL,
  `last_date_listener` datetime DEFAULT current_timestamp(),
  `last_sample_epoch_listener` bigint(20) unsigned NOT NULL DEFAULT 0,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_mysql_server_2` (`id_mysql_server`,`id_ts_file`),
  UNIQUE KEY `id_mysql_server` (`id_mysql_server`,`id_ts_file`,`date`),
  UNIQUE KEY `date` (`date`,`id_ts_file`,`id_mysql_server`),
  KEY `id_ts_file` (`id_ts_file`),
  KEY `id_mysql_server_4` (`id_mysql_server`,`date`),
  KEY `date_2` (`date`,`last_date_listener`),
  CONSTRAINT `ts_max_date_ibfk_3` FOREIGN KEY (`id_ts_file`) REFERENCES `ts_file` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=22859 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

`SHOW CREATE TABLE mysql_server;`

```
CREATE TABLE `mysql_server` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `is_deleted` tinyint(1) NOT NULL DEFAULT 0,
  `created_at` datetime NOT NULL DEFAULT current_timestamp(),
  `updated_at` datetime NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp(),
  `id_client` int(11) NOT NULL,
  `id_environment` int(11) NOT NULL,
  `name` varchar(100) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci NOT NULL,
  `display_name` varchar(100) NOT NULL,
  `ip` varchar(255) CHARACTER SET latin1 COLLATE latin1_bin NOT NULL,
  `hostname` varchar(200) NOT NULL DEFAULT '',
  `login` varchar(32) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci NOT NULL,
  `passwd` varchar(255) CHARACTER SET latin1 COLLATE latin1_bin NOT NULL,
  `database` varchar(64) NOT NULL,
  `is_password_crypted` int(11) NOT NULL,
  `is_ssl` int(11) NOT NULL DEFAULT 0,
  `port` int(11) NOT NULL,
  `ssh_nat` varchar(64) CHARACTER SET latin1 COLLATE latin1_bin NOT NULL DEFAULT '',
  `ssh_port` int(11) NOT NULL DEFAULT 22,
  `ssh_login` text CHARACTER SET latin1 COLLATE latin1_bin NOT NULL DEFAULT '\'\'',
  `is_sudo` int(11) NOT NULL DEFAULT 0,
  `is_root` int(11) NOT NULL DEFAULT 1,
  `is_monitored` int(11) NOT NULL DEFAULT 1,
  `monitoring_disabled_reason` enum('manual','license_quota') DEFAULT NULL,
  `is_proxy` int(11) NOT NULL DEFAULT 0,
  `is_vip` tinyint(1) NOT NULL DEFAULT 0,
  `is_binlog_relay` tinyint(1) NOT NULL DEFAULT 0,
  `is_clickhouse` tinyint(1) NOT NULL DEFAULT 0,
  `errant_gtid_set` text DEFAULT NULL COMMENT '#1280 P0.1 last computed errant GTID set
(canonical form, empty string = no errant)',
  `is_acknowledged` int(11) NOT NULL DEFAULT 0,
  `timeout` int(11) GENERATED ALWAYS AS (case when `is_proxy` = 1 then 11 else 1 end)
VIRTUAL,
  `errant_gtid_sampled_at` datetime DEFAULT NULL COMMENT '#1280 P0.1 wall-clock of the last
successful errant-set computation',
  `errant_gtid_ignore` tinyint(1) NOT NULL DEFAULT 0,
  `replica_lag_sla_seconds` int(11) NOT NULL DEFAULT 30,
  `is_out_of_service` tinyint(1) NOT NULL DEFAULT 0 COMMENT 'operator-flagged HS for reload
from master',
  PRIMARY KEY (`id`),
```

```
  UNIQUE KEY `name` (`name`),
  UNIQUE KEY `ip` (`ip`,`port`),
  KEY `id_client` (`id_client`),
  KEY `id_environment` (`id_environment`),
  KEY `is_monitored` (`id_client`,`is_monitored`,`is_deleted`) USING BTREE,
  KEY `is_deleted` (`is_deleted`)
) ENGINE=InnoDB AUTO_INCREMENT=337 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci WITH
SYSTEM VERSIONING
 PARTITION BY SYSTEM_TIME
PARTITIONS 2;
```

`SHOW CREATE TABLE ts_value_general_int;`

```
CREATE TABLE `ts_value_general_int` (
  `date` datetime NOT NULL,
  `id_ts_variable` int(11) NOT NULL,
  `id_mysql_server` int(11) NOT NULL,
  `value` bigint(20) unsigned DEFAULT NULL,
  PRIMARY KEY (`date`,`id_ts_variable`,`id_mysql_server`),
  KEY `id_mysql_server` (`id_mysql_server`,`id_ts_variable`,`date`)
) ENGINE=ROCKSDB DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci
 PARTITION BY RANGE (to_days(`date`))
(PARTITION `p740279` VALUES LESS THAN (740279) ENGINE = ROCKSDB,
 PARTITION `p740280` VALUES LESS THAN (740280) ENGINE = ROCKSDB,
 PARTITION `p740281` VALUES LESS THAN (740281) ENGINE = ROCKSDB,
 PARTITION `p740282` VALUES LESS THAN (740282) ENGINE = ROCKSDB,
 PARTITION `p740283` VALUES LESS THAN (740283) ENGINE = ROCKSDB,
 PARTITION `p740284` VALUES LESS THAN (740284) ENGINE = ROCKSDB,
 PARTITION `p740285` VALUES LESS THAN (740285) ENGINE = ROCKSDB,
 PARTITION `p740286` VALUES LESS THAN (740286) ENGINE = ROCKSDB,
 PARTITION `p740287` VALUES LESS THAN (740287) ENGINE = ROCKSDB,
 PARTITION `p740288` VALUES LESS THAN (740288) ENGINE = ROCKSDB,
 PARTITION `p740289` VALUES LESS THAN (740289) ENGINE = ROCKSDB,
 PARTITION `p740290` VALUES LESS THAN (740290) ENGINE = ROCKSDB,
 PARTITION `p740291` VALUES LESS THAN (740291) ENGINE = ROCKSDB,
 PARTITION `p740292` VALUES LESS THAN (740292) ENGINE = ROCKSDB,
 PARTITION `p740293` VALUES LESS THAN (740293) ENGINE = ROCKSDB,
 PARTITION `p740294` VALUES LESS THAN (740294) ENGINE = ROCKSDB,
 PARTITION `p740295` VALUES LESS THAN (740295) ENGINE = ROCKSDB,
 PARTITION `p740296` VALUES LESS THAN (740296) ENGINE = ROCKSDB,
 PARTITION `p740297` VALUES LESS THAN (740297) ENGINE = ROCKSDB,
```

```
 PARTITION `p740298` VALUES LESS THAN (740298) ENGINE = ROCKSDB,
 PARTITION `p740299` VALUES LESS THAN (740299) ENGINE = ROCKSDB,
 PARTITION `p740300` VALUES LESS THAN (740300) ENGINE = ROCKSDB,
 PARTITION `p740301` VALUES LESS THAN (740301) ENGINE = ROCKSDB,
 PARTITION `p740302` VALUES LESS THAN (740302) ENGINE = ROCKSDB,
 PARTITION `p740303` VALUES LESS THAN (740303) ENGINE = ROCKSDB,
 PARTITION `p740304` VALUES LESS THAN (740304) ENGINE = ROCKSDB,
 PARTITION `p740305` VALUES LESS THAN (740305) ENGINE = ROCKSDB,
 PARTITION `p740306` VALUES LESS THAN (740306) ENGINE = ROCKSDB,
 PARTITION `p740307` VALUES LESS THAN (740307) ENGINE = ROCKSDB,
 PARTITION `p740308` VALUES LESS THAN (740308) ENGINE = ROCKSDB,
 PARTITION `p740309` VALUES LESS THAN (740309) ENGINE = ROCKSDB,
 PARTITION `p740310` VALUES LESS THAN (740310) ENGINE = ROCKSDB);
```

Zmiana w kolektorze

Indeks unikalny tabeli `ts_variable` nosi nazwę `name` i obejmuje **(`name`, `from`)**. Kilka źródeł może używać tej samej nazwy metryki, a konkretna para wskazuje najwyżej jeden wiersz. Podczas zapisu źródło było znane. Odczyt filtrował po nazwie, typie i polu radical, ale pomijał źródło. Po wdrożeniu używa tej samej wartości co zapis. Poniżej ścieżka dostępności MySQL z parametrem identyfikatora serwera.

```
SELECT t.value AS previous_value
FROM ts_max_date AS m
JOIN ts_file AS f ON f.id = m.id_ts_file
JOIN ts_variable AS v ON v.id_ts_file = f.id
  AND v.name = 'mysql_available'
  AND v.`from` = 'mysql_server'  /* + */
  AND v.type = 'INT' AND v.radical = 'general'
JOIN mysql_server AS s ON s.id = m.id_mysql_server
  AND s.is_deleted = 0
LEFT JOIN ts_value_general_int AS t
  ON t.id_mysql_server = m.id_mysql_server
 AND t.id_ts_variable = v.id
 AND t.date = m.date
WHERE m.id_mysql_server = ? AND f.file_name = 'mysql_server'
LIMIT 1;
```

Oznaczony komentarzem wiersz to jedyna dodana część. `from` jest nazwą kolumny zapisaną w odwrotnych apostrofach. Dla tej ścieżki ma wartość `mysql_server`. Pozostałe kolektory przekazują własne źródło. Indeks istniał już przed wdrożeniem.

Źródło wyróżnia tylko 32 dodatkowe grupy wśród 3791 wierszy

Pogrupowaliśmy 3791 wierszy tabeli `ts_variable` w jednej migawce tylko do odczytu, 21 listopada 2026 r. o 21:25 czasu paryskiego. Wartość procentowa to iloraz liczby różnych grup i liczby wierszy tabeli.

Kolumny grupujące	Dokładna liczba grup	Grupy / wiersze	`SHOW INDEX` przed → po
(`name`)	3 759	99,16 %	3 841 → 3 840
(`name` , `from`)	3 791	100 %	3 841 → 3 840
(`from`)	42	1,11 %	42 → 42

Dwa pierwsze wiersze odpowiadają kolejnym prefiksom indeksu unikalnego `name`. Trzeci dotyczy odrębnego indeksu na `from`. Przy pierwszym pomiarze MariaDB podała **3841 dla obu pozycji** indeksu unikalnego. Jest to oszacowanie i może przekraczać rzeczywistą liczbę wierszy. Dokumentacja MariaDB o kardynalności.

Liczby wyjaśniają tę intuicję. **3729 nazw** występuje raz, **28** dwa razy, a **2** trzy razy. Grupowanie według (`name` , `from`) daje zaledwie **32 dodatkowe grupy** w całej tabeli. Stosunek liczby grup do liczby wierszy rośnie z **99,16% do 100%**, czyli o **0,84 punktu procentowego**. Dla metryki dostępności znika nawet ta różnica. Sama nazwa `mysql_available` wskazuje **jeden wiersz**, a podanie źródła wskazuje **ten sam wiersz**.

```
SELECT rows_per_name, COUNT(*) AS number_of_names
FROM (
    SELECT name, COUNT(*) AS rows_per_name
    FROM ts_variable
    GROUP BY name
) AS names
GROUP BY rows_per_name
ORDER BY rows_per_name;
```

Po odświeżeniu statystyk `ts_variable`

O 21:30 czasu paryskiego polecenie `ANALYZE NO_WRITE_TO_BINLOG TABLE pmacontrol.ts_variable` zwróciło `OK` na monitorowanej instancji. W `SHOW INDEX` wartość dla obu prefiksów spadła z 3841 do **3840**. Ponowne grupowanie bezpośrednio po odświeżeniu dało te same dokładne liczby grup.

InnoDB zapisuje oszacowania dla prefiksów w `mysql.innodb_index_stats`. Dla indeksu `name` wartość `n_diff_pfx01` wzrosła z 3757 do **3759**, a `n_diff_pfx02` z 3789 do **3791**. Obie odpowiadają teraz dokładnym wynikom. Przed odświeżeniem i po nim `EXPLAIN` szacował jeden wiersz dla obu osobnych odczytów z `ts_variable`. Typy dostępu pozostały takie same, `ref` dla nazwy i `const` dla pary.

Zapytania zliczające i wyniki źródłowe (ZIP)

Porównanie planów wykonania

Przed zmianą MariaDB używała indeksu `name` z dostępem `ref` po pierwszej kolumnie. Dwie równości określają cały klucz unikalny, więc dla `v` pojawia się `const`. Plany wykorzystują odpowiednio 102 i 168 bajtów klucza. Dostęp przez klucz główny do `t` również zmienia się z `eq_ref` na `const`. Liczba badanych wierszy na wykonanie całego zapytania spada z około trzech do jednego. Silnik może ustalić jednoznaczność przed dalszymi złączeniami.

Każda komórka podaje kolejność odczytu, typ dostępu, wybrany indeks i długość klucza w bajtach. Wiersze odpowiadają tym samym tabelom.

Tabela	Przed	Po
`ts_file` (f)	1 · `const` · `file_name` · 130 B	1 · `const` · `file_name` · 130 B
`ts_max_date` (m)	2 · `const` · `id_mysql_server_2` · 8 B	3 · `const` · `id_mysql_server_2` · 8 B
`ts_variable` (v)	3 · `ref` · `name` · 102 B	2 · `const` · `name` · 168 B
`mysql_server` (s)	4 · `ref` · `PRIMARY` · 4 B	5 · `ref` · `PRIMARY` · 4 B
`ts_value_general_int` (t)	5 · `eq_ref` · `PRIMARY` · 13 B	4 · `const` · `PRIMARY` · 13 B

W obu planach MariaDB szacuje po jednym wierszu dla każdej z pięciu tabel i wymienia te same 32 partycje dla `ts_value_general_int`.

Pełny plan przed zmianą (CSV) · Pełny plan po zmianie (CSV)

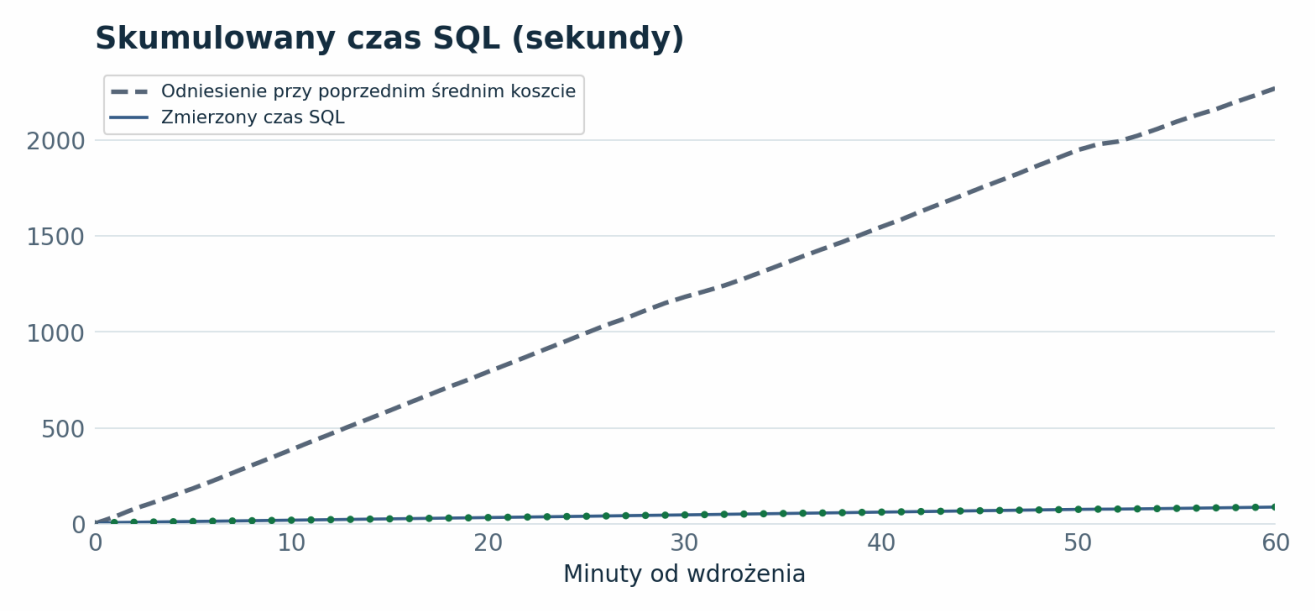
Zmiana czasu według etapu

Etap · mediana dla 50 serwerów	Przed	Po
Optymalizacja	35,085 ms	✓ **0,261 ms (÷134,27)**
Wykonanie po optymalizacji	0,107 ms	0,011 ms
Etap `Statistics`	54,983 ms	✓ **0,224 ms (÷245,46)**

Dwa pierwsze wiersze pochodzą z `ANALYZE FORMAT=JSON`. Etap `Statistics` zmierzono oddzielnie poleceniem `SHOW PROFILE`.

Przy samym `name` MariaDB **szacuje**, że odczyta jeden wiersz. Oba warunki równości wraz z unikalnym indeksem (`name` , `from`) **gwarantują** najwyżej jedno dopasowanie. MariaDB może więc traktować `v.id` jako stałą już podczas planowania złączenia z tabelą wartości. Na 50 serwerach mediana **czasu planowania spadła z 35,085 do 0,261 ms**. Czas późniejszego wykonania spadł z **0,107 do 0,011 ms**. Największą część zmierzonego zysku przyniosło planowanie złączeń.

Łączny czas SQL od wdrożenia



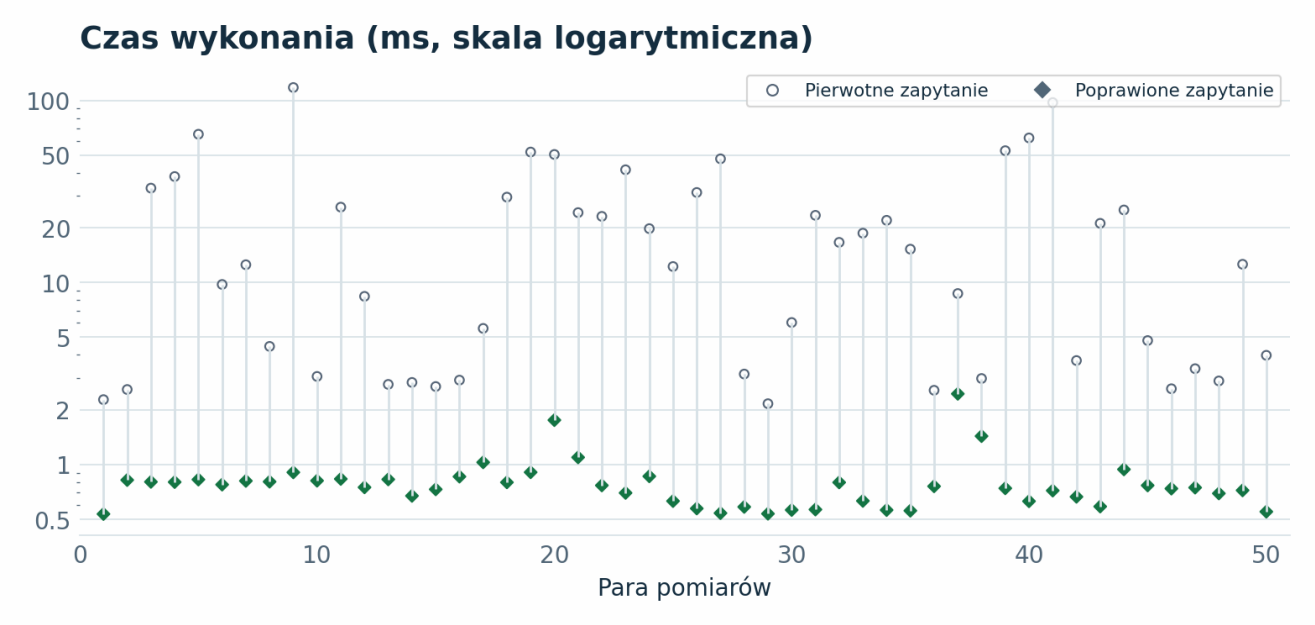
Obie krzywe zaczynają od zera i wykorzystują rzeczywistą liczbę wywołań po wdrożeniu. Linia ciągła sumuje zmierzony czas. Krzywa odniesienia stosuje do tych samych wywołań średni koszt z

poprzedniej godziny. Po godzinie to **86,5 s zmierzone** wobec **2271,4 s oszacowanych** według starego kosztu, czyli **szacowana oszczędność 36 min 25 s czasu SQL**. Czasy sesji mogą się nakładać i obejmują oczekiwanie. Estymacja nie mierzy minut CPU.

Sprawdzenie na 50 serwerach

W MariaDB 11.8.10 z wyłączoną pamięcią podręczną zapytań wylosowaliśmy bez zwracania 50 spośród 111 identyfikatorów dających wartość. Każda para odczytywała ten sam serwer w transakcji tylko do odczytu. Stara wersja była pierwsza 25 razy, nowa również 25. Czas zmierzono po stronie serwera przez `SHOW PROFILES`. **Wszystkie 50 wyników było zgodnych**, a nowa wersja była szybsza **w każdej parze**.

Wariant	Minimum (ms)	Mediana (ms)	Średnia (ms)	Maksimum (ms)
Przed	2,159	12,543	21,686	117,530
Po	✓ **0,535 (÷4,03)**	✓ **0,755 (÷16,62)**	✓ **0,804 (÷26,98)**	✓ **2,446 (÷48,05)**



Każda kolumna łączy dwa pomiary tego samego serwera. Skala logarytmiczna pozwala zobaczyć wartości poniżej 1 ms i maksimum 117,53 ms. Stosunek średnich w próbie wynosi 26,98, a średnich ważonych z dwóch godzin produkcyjnych 26,25.

Porównanie wariantów z wdrożonym zapytaniem

Każdy wiersz porównuje średnie czasy obu zapytań na tych samych serwerach w tej samej próbie. Każda seria obejmuje 50 serwerów. Liczba w nawiasie oznacza liczbę wykonań każdego wariantu.

Wariant	Odniesienie (ms)	Wariant (ms)
Tylko nazwa (150)	0,651	✗ **36,782 (×56,50)**
Tylko nazwa, wymuszony klucz główny tabeli wartości (150)	0,651	✗ **1,674 (×2,57)**
Pełny klucz, wymuszony klucz główny tabeli wartości (150)	0,651	0,630
Wymuszony indeks pokrywający w tabeli dat (50)	0,692	✗ **20,232 (×29,25)**
Aktywny serwer sprawdzany przez `EXISTS` (150)	0,651	0,663
Wartość z podzapytania skalarnego (150)	0,617	0,546
Wymuszona kolejność złączeń (50)	0,692	✗ **41,737 (×60,34)**
Podzapytanie skalarne z `EXISTS` (150)	0,617	0,563
Podzapytanie skalarne bez `LIMIT` (150)	0,617	0,560
Wdrożone złączenia bez `LIMIT` (150)	0,617	0,602
Podzapytanie skalarne z samą nazwą (150)	0,617	1,218

Zanonimizowane pomiary wariantów i etapów (ZIP)

Jak odtworzyć wykresy

Okno obejmuje 16:26:47–18:26:47 czasu paryskiego. Wartości na granicach przedziałów interpolujemy pomiędzy próbkami. Liczniki odczytane przy wdrożeniu wyznaczają punkt przejścia. Uwzględniamy 496 próbnych wywołań nowej wersji przed wdrożeniem oraz 224 końcowe wywołania starej po nim. Restart kolektorów około 18:17 wpływa na końcową część wykresu liczby wykonań.

Przyrost `COUNT_STAR` w każdej minucie daje bezpośrednio liczbę wywołań. Dla średniego opóźnienia dzielimy przyrost `SUM_TIMER_WAIT` przez przyrost `COUNT_STAR`, a następnie przez miliard, aby przeliczyć pikosekundy na milisekundy. Średnia godzinna wykorzystuje sumę czasu i liczbę wywołań. Pary pomiarów lepiej izolują efekt zmiany SQL niż dwie godziny obserwacji

systemu pod obciążeniem.

Dane i plany do pobrania

- Próbki liczników (CSV)
- Serie minutowe i suma (CSV)
- 50 anonimowych par (CSV)
- Szczegółowe plany (CSV)
- Dokumentacja EXPLAIN w MariaDB