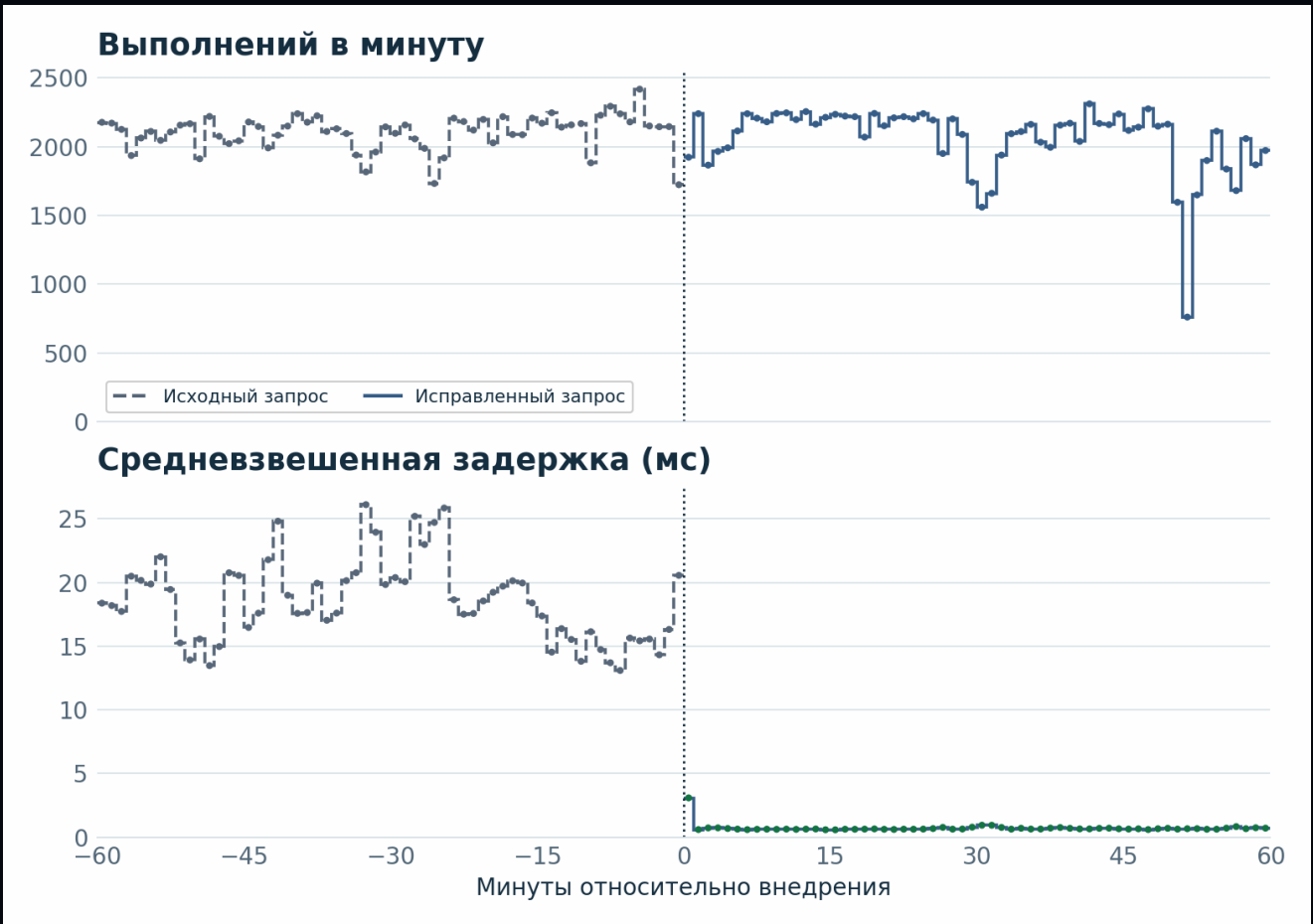
Почему время запроса MariaDB упало с 18,43 до 0,70 мс, хотя поиск по имени уже давал одну строку

Aurélien LEQUOY · November 21, 2026

PMACONTROL MARIADB PERFORMANCE SQL BENCHMARK OBSERVABILITY

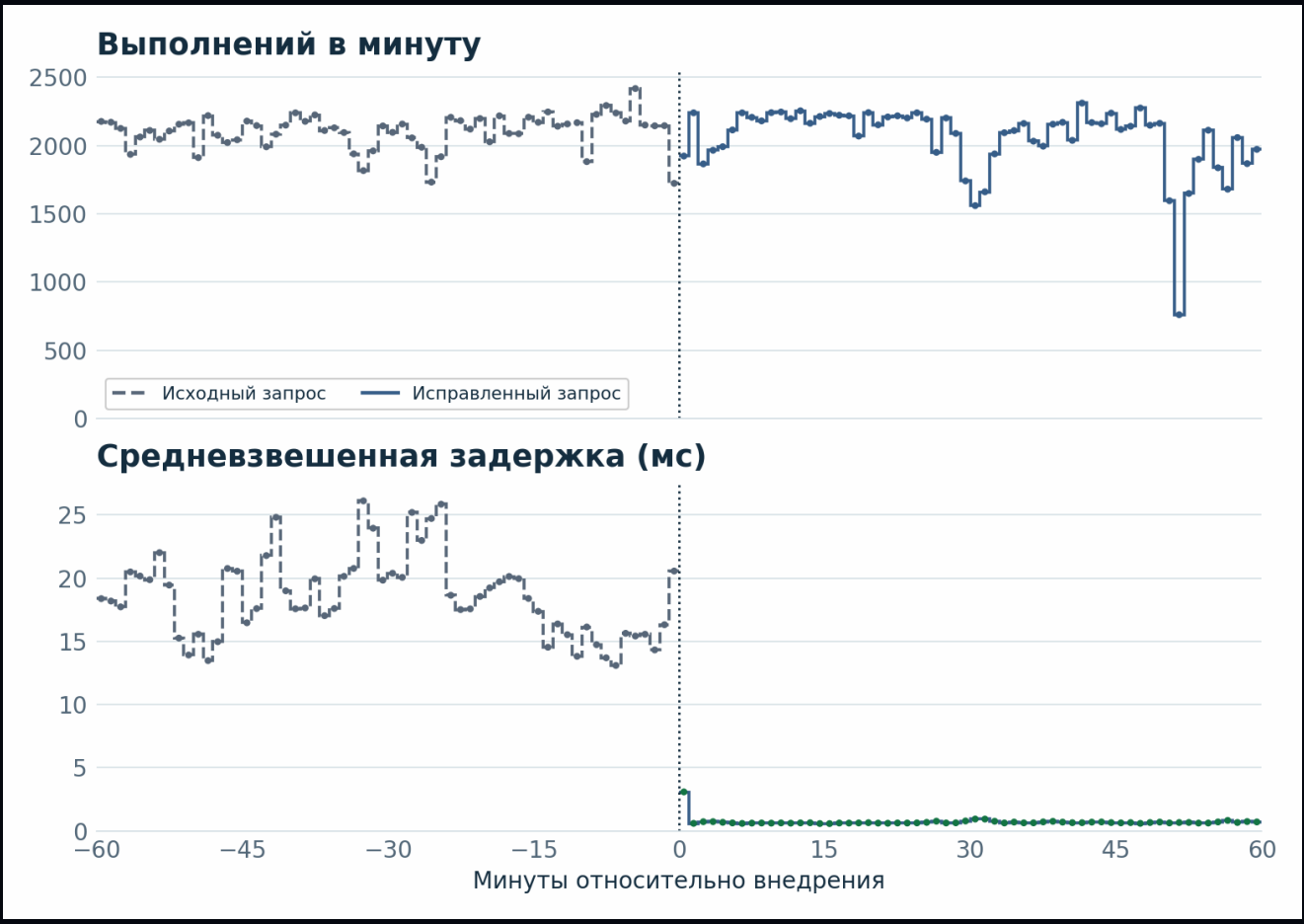


В `ts_variable` было **3 759 различных имён на 3 791 строку**, так что столбец `name` уже обладал очень высокой избирательностью. Для `mysql_available` находилась ровно одна строка. Условие `v.\`from\` = 'mysql_server'` оставляло ту же строку. Казалось бы, ускорение должно быть незначительным. Однако при нагрузке около 2 100 вызовов в минуту **среднее время запроса в рабочей системе упало с 18,43 до 0,70 мс**. Что выиграла MariaDB, если второе условие не отсеяло ни одной дополнительной строки для этой метрики?

Перед записью доступности сервиса PmaControl читает предыдущее значение, чтобы обнаружить смену состояния. За час до внедрения эти чтения заняли **2326 секунд суммарного времени SQL**. За следующий час они заняли **87 секунд** при близком числе вызовов. Изменилось одно условие соединения.

Два часа вокруг внедрения

21 ноября 2026 года в **17:26:47 по Парижу** сборщик начал использовать изменённый запрос. После изменения текста SQL Performance Schema присвоила ему новый digest. Если следить только за старым, число выполнений падает до нуля. Чтобы увидеть работу сервиса, мы складываем приращения счётчиков обеих версий за минутные интервалы.



В каждом ряду 120 точек, по одной на минуту. Верхний график показывает число вызовов в каждую минуту. Нижний показывает среднюю задержку тех же вызовов. Вертикальный пунктир отмечает внедрение. Вызовы другой версии вблизи перехода учтены.

В таблицах зелёный цвет (✓) обозначает улучшение, а красный (✗) — ухудшение, если большее значение строго больше удвоенного меньшего. Для длительностей разница

также должна составлять не менее 1 мс. Число вызовов характеризует нагрузку. Порог помогает читать результаты. Линии на графиках различают периоды. Каждую точку задержки после внедрения сравнивают со средневзвешенной задержкой предыдущего часа. Для накопленного времени сравнивают одинаковый объём вызовов, а в парном тесте используют исходное время каждой пары.

Показатель	**Час до**	**Час после**
Оценка числа вызовов	126 229	123 241
Вызовов в минуту	2 104	2 054
Средневзвешенная задержка	18,43 ms	✓ **0,70 ms (÷26,25)**
Суммарное время SQL, с	2 326,5	✓ **86,5 (÷26,88)**
Вызовов на секунду SQL	54	✓ **1 424 (×26,25)**
Просмотрено строк на вызов	2,99	✓ **1,00 (÷2,98)**

Эффективность в пересчёте на секунду суммарного времени SQL выросла с **54 до 1424 чтений**. Задержка снизилась на **96,2%**, а наблюдаемая частота выполнения лишь на **2,4%**. Эффективность отражает стоимость чтения. Фактическая частота также зависит от периода опроса и параллелизма.

Что именно читает запрос

Для заданного сервера `ts_max_date` хранит последнюю дату измерений файла. `ts_file` определяет сам файл, а `ts_variable` — метрику. Таблица `mysql_server` исключает удалённые серверы. Внешнее соединение ищет целое значение для того же времени, метрики и сервера. Если измерение отсутствует, результат этого соединения может быть `NULL`.

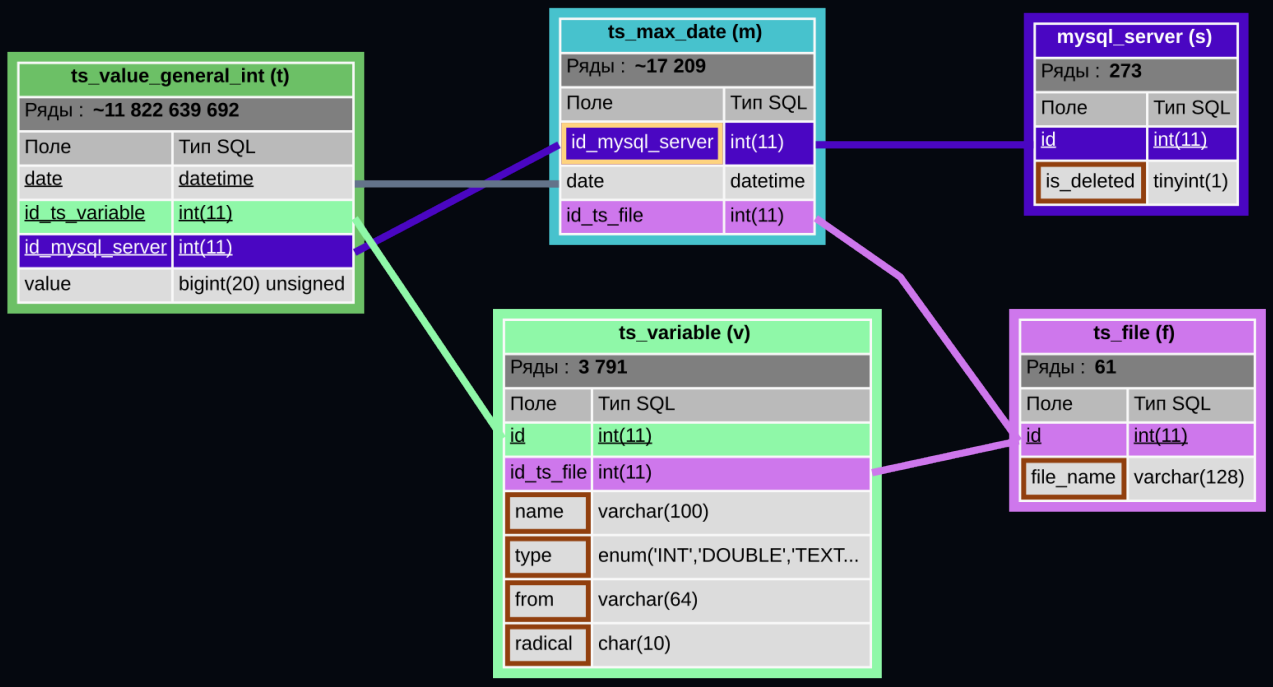


Схема исправленного запроса из PmaControl. Стрелки соединяют столбцы, участвующие в соединениях. Коричневой рамкой отмечены поля с условиями фильтрации, включая `ts_variable.from`. Знак `~` перед числом строк обозначает оценку на момент выгрузки.

Определения пяти таблиц

Определения получены командой `SHOW CREATE TABLE` на исследуемом экземпляре MariaDB. В определении `ts_variable` строка `UNIQUE KEY` показывает два столбца, которые использует исправленный запрос, в порядке их следования в индексе.

`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);
```

Изменение в сборщике

Уникальный индекс таблицы `ts_variable` называется `name` и включает **(`name`, `from`)**. Имя метрики может встречаться у нескольких источников, но для конкретной пары здесь существует не более одной строки. При записи источник уже был известен. При чтении фильтр учитывал имя, тип и поле radical, упуская источник. Новый код использует то же значение источника, что и при записи. Ниже запрос доступности MySQL с параметром для идентификатора сервера.

```
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;
```

Единственное добавление отмечено комментарием. Здесь `from` — имя столбца в обратных кавычках. Для этого вида проверки его значение равно `mysql_server`. Другие сборщики передают собственный источник. Индекс существовал до внедрения.

Источник добавил всего 32 группы на 3 791 строку

Мы сгруппировали 3 791 строку `ts_variable` в одном снимке данных в транзакции только для чтения 21 ноября 2026 года в 21:25 по парижскому времени. Процент в таблице равен отношению числа различных групп к числу строк.

Группировка	Точное число групп	Группы / строки	`SHOW INDEX` до → после
(`name`)	3 759	99,16 %	3 841 → 3 840
(`name` , `from`)	3 791	100 %	3 841 → 3 840
(`from`)	42	1,11 %	42 → 42

Первые две строки соответствуют последовательным префиксам уникального индекса `name`. Третья относится к отдельному индексу по `from`. При первом замере MariaDB указала **3 841 для обеих позиций** уникального индекса. Это оценка, и она может превышать фактическое число строк. Справка MariaDB по кардинальности.

Подсчёт объясняет исходное ожидание. **3 729 имён** встречаются один раз, **28** дважды, **2** трижды. Группировка по (`name` , `from`) добавляет лишь **32 группы** во всей таблице. Отношение числа групп к числу строк растёт с **99,16 % до 100 %**, всего на **0,84 процентного пункта**. Для метрики доступности исчезает даже эта разница. Имя `mysql_available` находит **одну строку**, и при указании источника находится **та же строка**.

```
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;
```

После обновления статистики `ts_variable`

В 21:30 по парижскому времени команда `ANALYZE NO_WRITE_TO_BINLOG TABLE pmacontrol.ts_variable` вернула `OK` на исследуемом сервере. Значение `SHOW INDEX` для обоих префиксов изменилось с 3 841 до **3 840**. Повторная группировка сразу после обновления дала те же точные числа групп.

InnoDB хранит оценки по префиксам в `mysql.innodb_index_stats`. Для индекса `name` значение `n_diff_pfx01` выросло с 3 757 до **3 759**, а `n_diff_pfx02` — с 3 789 до **3 791**. Теперь оба совпадают с точным подсчётом. До и после обновления `EXPLAIN` оценивал каждый из двух отдельных запросов к `ts_variable` в одну строку. Тип доступа остался `ref` при поиске по имени и `const` при поиске по паре.

Запросы для подсчёта и исходные результаты (ZIP)

Сравнение планов выполнения

Раньше MariaDB находила строки по первому полю индекса `name` методом `ref`. Два равенства задают полный уникальный ключ, что даёт доступ `const` к `v`. Длина использованной части ключа в планах составила 102 и 168 байт. Доступ к `t` по первичному ключу также изменился с `eq_ref` на `const`. Число просмотренных строк для всего запроса упало примерно с трёх до одной. Движок устанавливает уникальность до дальнейших соединений.

В каждой ячейке указаны порядок чтения, тип доступа, выбранный индекс и длина ключа в байтах. В одной строке показана одна и та же таблица.

Таблица	До	После
`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

В обоих планах MariaDB оценивает число строк в каждой из пяти таблиц как единицу и перечисляет одни и те же 32 раздела для `ts_value_general_int`.

Полный план до изменения (CSV) · Полный план после изменения (CSV)

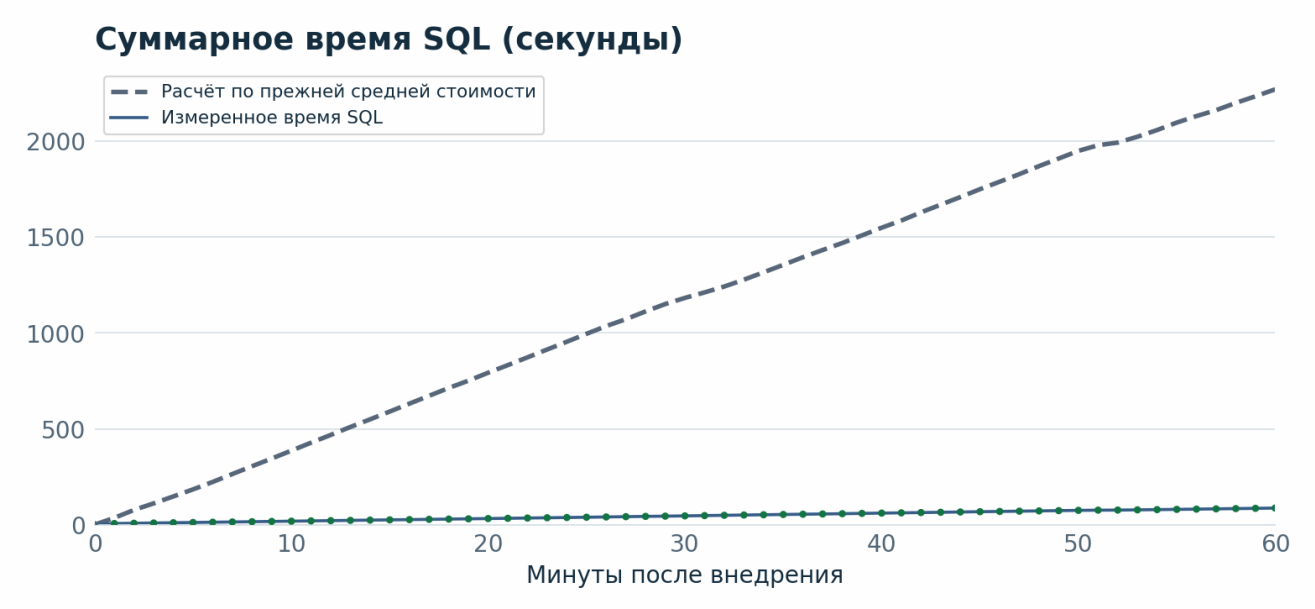
Как изменилось время по этапам

Этап · медиана по 50 серверам	До	После
Оптимизация	35,085 ms	✓ **0,261 ms (÷134,27)**
Выполнение после оптимизации	0,107 ms	0,011 ms
Этап `Statistics`	54,983 ms	✓ **0,224 ms (÷245,46)**

Первые две строки получены с помощью `ANALYZE FORMAT=JSON`. Этап `Statistics` измерен отдельно с помощью `SHOW PROFILE`.

По одному `name` MariaDB **оценивает** результат в одну строку. При двух равенствах уникальный индекс (`name`, `from`) **гарантирует** не более одного совпадения. Поэтому уже при планировании соединения с таблицей значений MariaDB может считать `v.id` константой. На 50 серверах медиана **времени планирования снизилась с 35,085 до 0,261 мс**, а последующего выполнения — с **0,107 до 0,011 мс**. Основная часть измеренного выигрыша пришлась на построение плана соединений.

Суммарное время SQL с момента внедрения

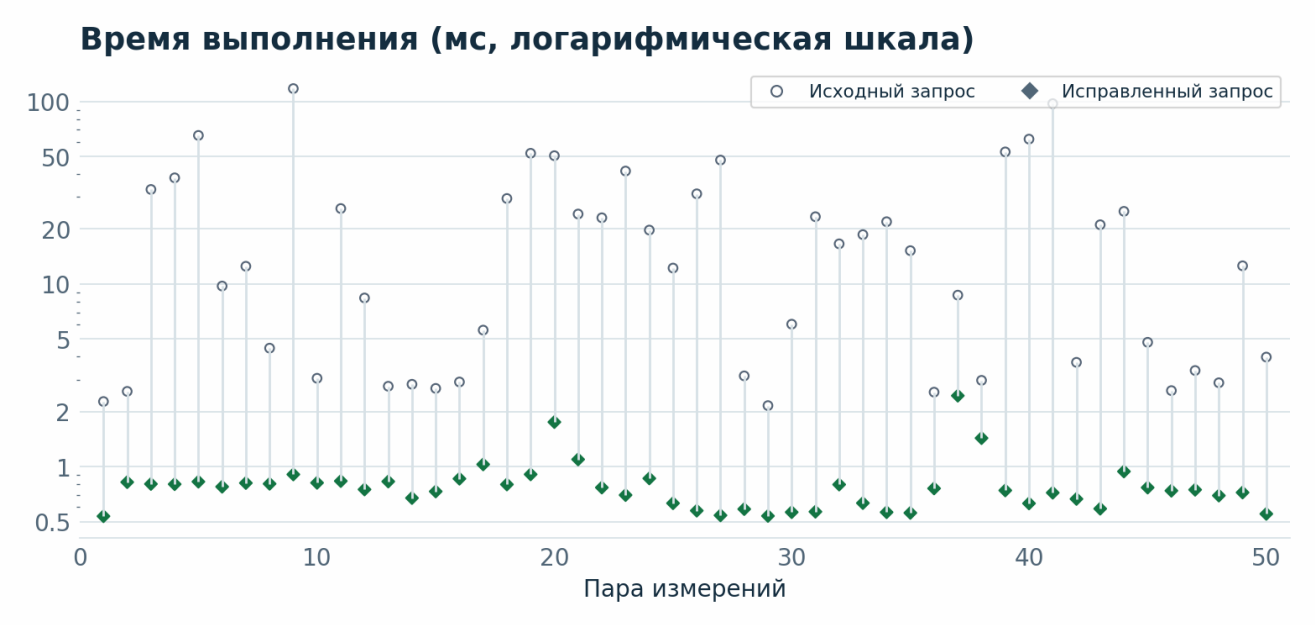


Обе кривые начинаются с нуля и используют одно и то же фактическое число вызовов после внедрения. Сплошная линия накапливает измеренное время. Для расчётной кривой к тем же вызовам применяется средняя стоимость предыдущего часа. Через час **измерено 86,5 с**, а **оценка при прежней стоимости равна 2271,4 с**. Расчётная экономия составляет **36 мин 25 с времени SQL**. Время разных соединений может перекрываться и включает ожидание. Это не измерение минут CPU.

Проверка на 50 серверах

В MariaDB 11.8.10 с отключённым кешем запросов мы случайно выбрали 50 различных идентификаторов из 111, для которых нашлось значение. Каждая пара читала один сервер в транзакции только для чтения. Исходный запрос выполнялся первым в 25 парах, исправленный — в остальных 25. Время получено на сервере через `SHOW PROFILES`. **Результаты совпали во всех 50 парах**, и исправленная версия **каждый раз была быстрее**.

Вариант	Минимум (ms)	Медиана (ms)	Среднее (ms)	Максимум (ms)
До	2,159	12,543	21,686	117,530
После	✓ **0,535 (÷4,03)**	✓ **0,755 (÷16,62)**	✓ **0,804 (÷26,98)**	✓ **2,446 (÷48,05)**



Каждый столбец соединяет два замера одного сервера. Логарифмическая шкала показывает и значения менее 1 мс, и максимум 117,53 мс. Отношение средних в выборке

составило 26,98, а отношение взвешенных средних за два часа работы — 26,25.

Сравнение вариантов с внедрённым запросом

Каждая строка сравнивает среднее время обоих запросов на одних и тех же серверах в одном испытании. Каждая серия охватывает 50 серверов. В скобках указано число выполнений каждого варианта.

Вариант	Базовый запрос (мс)	Вариант (мс)
Только имя (150)	0,651	**✗ 36,782 (×56,50)**
Только имя, принудительный первичный ключ таблицы значений (150)	0,651	**✗ 1,674 (×2,57)**
Полный ключ, принудительный первичный ключ таблицы значений (150)	0,651	0,630
Принудительный покрывающий индекс таблицы дат (50)	0,692	**✗ 20,232 (×29,25)**
Проверка активного сервера через `EXISTS` (150)	0,651	0,663
Чтение значения скалярным подзапросом (150)	0,617	0,546
Фиксированный порядок соединений (50)	0,692	**✗ 41,737 (×60,34)**
Скалярный подзапрос с `EXISTS` (150)	0,617	0,563
Скалярный подзапрос без `LIMIT` (150)	0,617	0,560
Внедрённые соединения без `LIMIT` (150)	0,617	0,602
Скалярный подзапрос только по имени (150)	0,617	1,218

Обезличенные измерения вариантов и этапов (ZIP)

Как воспроизвести графики

Окно охватывает время с 16:26:47 до 18:26:47 по Парижу. На границах интервалов счётчики интерполированы между снимками. Счётчики, прочитанные в момент внедрения, закрепляют точку перехода. В расчёт входят 496 тестовых вызовов новой версии до внедрения и 224 последних вызова старой после него. Перезапуск сборщиков около 18:17 также повлиял на конец графика частоты.

Приращение `COUNT_STAR` за одну минуту непосредственно даёт число вызовов за эту минуту. Для средней задержки делим приращение `SUM_TIMER_WAIT` на приращение `COUNT_STAR`, затем на миллиард для перевода пикосекунд в миллисекунды. Часовая средняя использует суммарное время и число вызовов. Парные замеры точнее выделяют эффект изменения SQL, чем два часа наблюдений под нагрузкой.

Данные и планы для скачивания

- Снимки счётчиков (CSV)
- Минутные и накопительные данные (CSV)
- 50 обезличенных пар (CSV)
- Подробные планы (CSV)
- Документация MariaDB по EXPLAIN