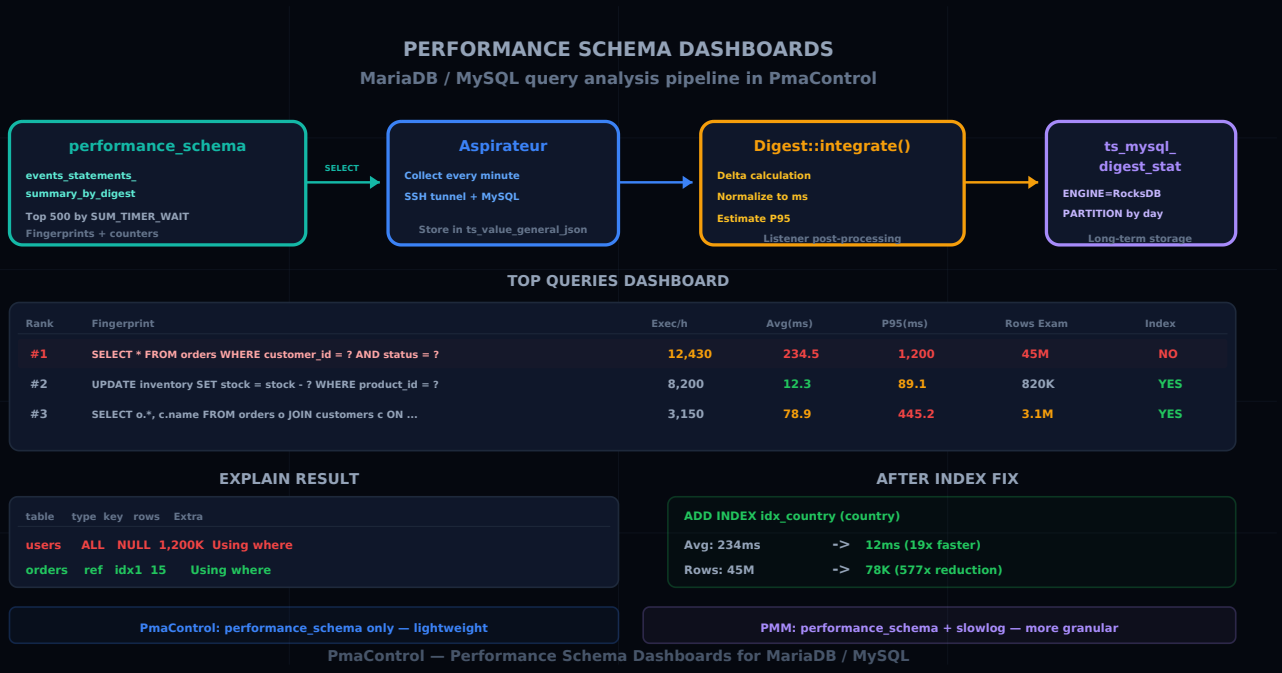
# Performance Schema i dashboardy PmaControl: tropienie wolnych zapytań

**Aurélien LEQUOY** · April 13, 2026

MARIADB MYSQL PERFORMANCE-SCHEMA DIAGNOSTICS PMACONTROL




## Performance Schema: nieeksploatowana kopalnia złota

`performance_schema` jest domyślnie włączone w MariaDB / MySQL od lat. A mimo to większość DBA nie eksploatuje go na co dzień. Powód jest prosty: surowe dane są trudne do odczytania. Dziesiątki tabel, miliony wierszy, kumulatywne liczniki — bez narzędzia agregującego to jest szum.

PmaControl transformuje ten szum w sygnał. Zbiera dane z `performance_schema` za pomocą Aspirateur, agreguje je przez Listener (Digest::integrate) i prezentuje na wykorzystywalnych dashboardach. Ten artykuł opisuje kompletny pipeline, od źródła do dashboardu.

## Sprawdzenie, czy performance_schema jest włączone

### MariaDB

```
SHOW GLOBAL VARIABLES LIKE 'performance_schema';
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| performance_schema | ON    |
+--------------------+-------+
```

Jeśli `OFF`, dodaj do pliku konfiguracyjnego:

```
[mysqld]
performance_schema = ON
```

Restart jest wymagany — ta zmienna nie jest dynamiczna.

### MySQL

To samo na MySQL. Zmienna jest tylko do odczytu i wymaga restartu:

```
[mysqld]
performance_schema = ON
```

### Wpływ na wydajność

Klasyczne pytanie: "czy performance_schema spowalnia mój serwer?" Odpowiedź w 2026 roku brzmi: **nie, w mierzalny sposób**. Narzut wynosi rzędu 1-3% przy typowych obciążeniach. Korzyść w postaci widoczności znacznie przewyższa ten koszt.

Jedyny wyjątek: serwery z ekstremalnym obciążeniem (>100 000 zapytań/sekundę), gdzie liczy się każdy procent. W takim przypadku dezaktywuj niepotrzebne instrumenty zamiast całego performance_schema.

## Źródło: events_statements_summary_by_digest

Kluczowa tabela, którą PmaControl eksploatuje, to:

```
SELECT * FROM performance_schema.events_statements_summary_by_digest
ORDER BY SUM_TIMER_WAIT DESC
LIMIT 10\G
```

Ta tabela zawiera podsumowanie według **fingerprinta** (znormalizowanego odcisku) każdego wykonanego zapytania. Oto najważniejsze kolumny:

| Kolumna | Opis |
|---|---|
| `DIGEST` | Unikalny hash fingerprinta |
| `DIGEST_TEXT` | Znormalizowany tekst zapytania (parametry zastąpione przez `?`) |
| `COUNT_STAR` | Łączna liczba wykonań |
| `SUM_TIMER_WAIT` | Łączny czas wykonania (w pikosekundach) |
| `AVG_TIMER_WAIT` | Średni czas na wykonanie |
| `SUM_ROWS_EXAMINED` | Łączna liczba zbadanych wierszy |
| `SUM_ROWS_SENT` | Łączna liczba zwróconych wierszy |
| `FIRST_SEEN` | Pierwsze wykonanie |
| `LAST_SEEN` | Ostatnie wykonanie |

Fingerprint to kamień węgielny: normalizuje `SELECT * FROM users WHERE id = 42` i `SELECT * FROM users WHERE id = 1337` w jeden fingerprint `SELECT * FROM users WHERE id = ?`. Pozwala to na agregowanie statystyk niezależnie od wartości parametrów.

## Pipeline PmaControl

### Krok 1: Zbieranie przez Aspirateur

Aspirateur okresowo wykonuje następujące zapytanie na każdym nadzorowanym serwerze:

```
SELECT
    DIGEST,
    DIGEST_TEXT,
    COUNT_STAR,
    SUM_TIMER_WAIT,
    AVG_TIMER_WAIT,
    SUM_ROWS_EXAMINED,
    SUM_ROWS_SENT,
    SUM_NO_INDEX_USED,
    SUM_NO_GOOD_INDEX_USED,
```

```
    FIRST_SEEN,
    LAST_SEEN
FROM performance_schema.events_statements_summary_by_digest
WHERE DIGEST IS NOT NULL
ORDER BY SUM_TIMER_WAIT DESC
LIMIT 500;
```

`LIMIT 500` jest celowy: PmaControl koncentruje się na 500 zapytaniach najkosztowniejszych pod względem łącznego czasu. Szybkie i rzadkie zapytania nie są interesujące z punktu widzenia optymalizacji.

Wyniki są przechowywane w `ts_value_general_json` ze znacznikiem czasu.

### Krok 2: Przetwarzanie przez Listener

Gdy Listener wykryje nowe dane digest, uruchamia `Digest::integrate()`. Ta funkcja:

1. **Oblicza delty**: ponieważ `performance_schema` dostarcza kumulatywne liczniki (od ostatniego `TRUNCATE` lub restartu), Digest::integrate oblicza różnicę między dwoma zbiorami, aby uzyskać metryki z danego okresu.
2. **Normalizuje czasy**: pikosekundy są konwertowane na milisekundy do wyświetlania.
3. **Oblicza percentyle**: P95 (95. percentyl) czasu wykonania jest szacowany na podstawie rozkładów.
4. **Zapisuje do ts_mysql_digest_stat**: dedykowanej tabeli statystyk digest, partycjonowanej dziennie i wykorzystującej silnik RocksDB do kompresji.

```
Aspirateur
  → SELECT FROM performance_schema (every minute)
  → INSERT INTO ts_value_general_json

Listener
  → Detect new data (ts_max_date changed)
  → Digest::integrate()
     → Delta calculation (current - previous)
     → Normalize to milliseconds
     → Estimate P95
     → INSERT INTO ts_mysql_digest_stat
```

### Tabela ts_mysql_digest_stat

To długoterminowe przechowywanie statystyk digest:

```
CREATE TABLE ts_mysql_digest_stat (
    id BIGINT UNSIGNED AUTO_INCREMENT,
    server_id INT UNSIGNED,
    digest VARCHAR(64),
    digest_text TEXT,
    period_start DATETIME,
    period_end DATETIME,
    exec_count BIGINT UNSIGNED,
    total_time_ms DECIMAL(20,3),
    avg_time_ms DECIMAL(15,3),
    p95_time_ms DECIMAL(15,3),
    rows_examined BIGINT UNSIGNED,
    rows_sent BIGINT UNSIGNED,
    no_index_used BIGINT UNSIGNED,
    PRIMARY KEY (id),
    KEY idx_server_digest (server_id, digest, period_start)
) ENGINE=ROCKSDB
PARTITION BY RANGE (TO_DAYS(period_start)) (
    PARTITION p20260413 VALUES LESS THAN (TO_DAYS('2026-04-14')),
    PARTITION p20260414 VALUES LESS THAN (TO_DAYS('2026-04-15')),
    ...
);
```

Partycjonowanie dzienne umożliwia:

- Szybkie czyszczenie: `ALTER TABLE ts_mysql_digest_stat DROP PARTITION p20260401;`
- Szybkie zapytania na zakresie dat
- Precyzyjną kontrolę retencji

## Dashboardy

### Widok Top Queries

Główny dashboard wyświetla najkosztowniejsze zapytania, posortowane według łącznego czasu:

| Rang | Fingerprint | Exec/h | Avg(ms) | P95(ms) | Rows Exam |
|---|---|---|---|---|---|
| 1 | SELECT * FROM orders WHERE customer_id = ? AND status = ? | 12,430 | 45.2 | 234.5 | 1,245,000 |
| 2 | UPDATE inventory SET stock = stock - ? WHERE product_id = ? | 8,200 | 12.3 | 89.1 | 820,000 |
| 3 | SELECT o.*, c.name FROM orders o JOIN customers c ON o.customer_id = c.id | 3,150 | 78.9 | 445.2 | 3,150,000 |
| 4 | INSERT INTO audit_log (...) | 45,600 | 1.2 | 5.3 | 0 |
| 5 | SELECT COUNT(*) FROM sessions WHERE last_active < ? | 980 | 234.5 | 890.1 | 98,000,000 |

Każdy wiersz jest klikalny, aby przejść do szczegółów.

## Widok szczegółowy fingerprinta

Po kliknięciu fingerprinta PmaControl wyświetla:

- **Pełny tekst** znormalizowanego zapytania
- **Historię**: ewolucję średniego czasu wykonania i P95 przez ostatnie 30 dni
- **Współczynnik** rows_examined / rows_sent — wysoki współczynnik (>100:1) wskazuje na skanowanie tabeli lub brakujący indeks
- **Flagę no_index_used** — ile wykonań nie użyło żadnego indeksu

## Identyfikacja brakujących indeksów

Współczynnik rows_examined / rows_sent to najpotężniejszy wskaźnik. Weźmy przykład:

```
Fingerprint: SELECT * FROM orders WHERE customer_id = ?
Rows examined: 1,245,000 (łącznie)
Rows sent: 12,430 (łącznie)
Współczynnik: 100:1
```

Ten współczynnik 100:1 oznacza, że MariaDB / MySQL bada 100 wierszy, aby zwrócić 1. To klasyczny objaw full table scan lub nieskutecznego indeksu.

Działanie: sprawdź obecność indeksu na `customer_id`:

```
SHOW INDEX FROM orders WHERE Column_name = 'customer_id';
```

Jeśli indeks nie istnieje:

```
ALTER TABLE orders ADD INDEX idx_customer_id (customer_id);
```

### Flaga SUM_NO_INDEX_USED

PmaControl wyświetla na czerwono zapytania, w których `SUM_NO_INDEX_USED` jest wysoki. Ta flaga jest aktywowana, gdy MariaDB / MySQL wykonuje full table scan — to często problem wydajnościowy numer jeden.

### EXPLAIN z PmaControl

Dla zapytań zidentyfikowanych jako problematyczne PmaControl może wykonać `EXPLAIN` bezpośrednio:

```
EXPLAIN SELECT * FROM orders WHERE customer_id = 42 AND status = 'pending';
```

Wynik jest wyświetlany z kodem kolorystycznym:

- **Zielony**: `type = ref` lub `type = eq_ref` — użycie indeksu, dobrze
- **Bursztynowy**: `type = range` — skanowanie zakresu, akceptowalne
- **Czerwony**: `type = ALL` — full table scan, do naprawienia

## Integracja z agentem Marina+

Marina+ to agent automatycznej optymalizacji PmaControl. Analizuje dane digest i proponuje sugestie:

1. **Brakujące indeksy**: wykrywa zapytania z wysokim współczynnikiem rows_examined/rows_sent i sugeruje indeksy do utworzenia
2. **Zapytania do przepisania**: identyfikuje nieskuteczne wzorce (SELECT *, skorelowane podzapytania, ORDER BY na nieindeksowanej kolumnie)
3. **Konfiguracja**: dostosowuje parametry serwera na podstawie wzorców zapytań (sort_buffer_size, join_buffer_size, itd.)

Marina+ nie modyfikuje niczego automatycznie — generuje rekomendacje, które DBA weryfikuje i stosuje.

## Porównanie z PMM (Percona Monitoring and Management)

PMM i PmaControl eksploatują to samo źródło danych ( `performance_schema` ), ale z różnymi podejściami:

| Aspekt | PmaControl | PMM |
|---|---|---|
| Źródło | performance_schema | performance_schema + slowlog |
| Agent | Aspirateur (SSH + MySQL) | mysqld_exporter + QAN |
| Przechowywanie | ts_mysql_digest_stat (RocksDB) | ClickHouse (QAN) |
| Fingerprinting | Po stronie serwera (MariaDB / MySQL natywny) | Po stronie klienta (agent Percona) |
| P95 | Szacowany na podstawie rozkładów | Obliczany ze slowloga |
| Historia | Partycjonowana dziennie, konfigurowalna retencja | ClickHouse, konfigurowalna retencja |
| Działania | Zintegrowany EXPLAIN, sugestie Marina+ | Query Analytics + PMM UI |

Główna różnica: **PMM łączy performance_schema i slowlog** dla dokładniejszych percentyli. PmaControl bazuje wyłącznie na performance_schema, co jest lżejsze, ale mniej granularne.

Zaleta PmaControl: integracja z resztą ekosystemu (replikacja, topologia, alerty, akcje). PMM jest lepszy do czystej analizy zapytań, PmaControl jest lepszy do globalnego oglądu infrastruktury.

## Przypadek praktyczny: znajdowanie i naprawianie wolnego zapytania

Scenariusz: dashboard PmaControl pokazuje zapytanie, które pochłania 40% łącznego czasu serwera.

### Krok 1: Identyfikacja

W dashboardzie Top Queries:

```
#1  SELECT u.*, p.* FROM users u
    JOIN purchases p ON u.id = p.user_id
    WHERE p.created_at > ? AND u.country = ?
```

```
    Exec/h: 5,200   Avg: 234ms   P95: 1,200ms   Rows exam: 45M
```

### Krok 2: Analiza

Współczynnik rows_examined / rows_sent jest katastrofalny: 45 milionów zbadanych wierszy na ~5 200 wyników na godzinę.

EXPLAIN z PmaControl:

```
+----+------+----------+------+------+------+----------+
| id | type | table    | key  | rows | filt | Extra    |
+----+------+----------+------+------+------+----------+
|  1 | ALL  | users    | NULL | 1.2M | 10%  | where    |
|  1 | ref  | purchases| idx1 | 15   | 33%  | where    |
+----+------+----------+------+------+------+----------+
```

Problem: `users` jest skanowany w trybie full table ( `type = ALL` ). Nie ma indeksu na `country` .

### Krok 3: Naprawa

```
ALTER TABLE users ADD INDEX idx_country (country);
```

### Krok 4: Weryfikacja

Po dodaniu indeksu dashboard PmaControl pokazuje poprawę w ciągu godziny:

```
#1  SELECT u.*, p.* FROM users u
    JOIN purchases p ON u.id = p.user_id
    WHERE p.created_at > ? AND u.country = ?

    Exec/h: 5,200   Avg: 12ms   P95: 45ms   Rows exam: 78K
```

Średni czas spadł z 234ms do 12ms (x19), a liczba zbadanych wierszy z 45M do 78K (x577).

## Dobre praktyki

### 1. Nie wykonuj TRUNCATE performance_schema ręcznie

PmaControl oblicza delty między dwoma zbiorami. Jeśli wykonasz `TRUNCATE TABLE performance_schema.events_statements_summary_by_digest`, liczniki wracają do zera i pierwsza delta będzie nieprawidłowa. Zostaw zarządzanie PmaControl.

### 2. Zwiększ performance_schema_digests_size, jeśli konieczne

Domyślnie MariaDB / MySQL przechowuje N pierwszych fingerprintów. Jeśli twoja aplikacja ma więcej odrębnych zapytań niż limit, najrzadsze są usuwane:

```
[mysqld]
performance_schema_digests_size = 10000  ; domyślnie ~5000
```

### 3. Koreluj ze slow query logiem

PmaControl przez performance_schema daje "co" (jakie zapytania są wolne). Slow query log daje "kiedy" (w jakim dokładnie momencie). Te dwa źródła się uzupełniają.

### 4. Monitoruj współczynnik rows_examined / rows_sent

To najłatwiej wdrażalny wskaźnik. Współczynnik > 100:1 to prawie zawsze brakujący indeks. Współczynnik > 1000:1 to pilny problem.

### 5. Używaj P95, nie średniej

Średnia maskuje wartości odstające. Zapytanie ze średnią 10ms, ale P95 wynoszącym 500ms ma problem sporadyczny (lock contention, zimny cache, niestabilny plan wykonania). P95 ujawnia te problemy.

## Podsumowanie

Performance_schema to najlepsze źródło danych do optymalizacji zapytań MariaDB / MySQL. PmaControl automatyzuje zbieranie, agregację i prezentację tych danych przez pipeline Aspirateur → Digest::integrate → ts_mysql_digest_stat → dashboard.

Rezultat: ciągła widoczność najkosztowniejszych zapytań, brakujących indeksów i ewolucji wydajności w czasie. W połączeniu z Marina+ do zautomatyzowanych sugestii, to kompletny workflow optymalizacji wydajności — od wykrycia do naprawy.