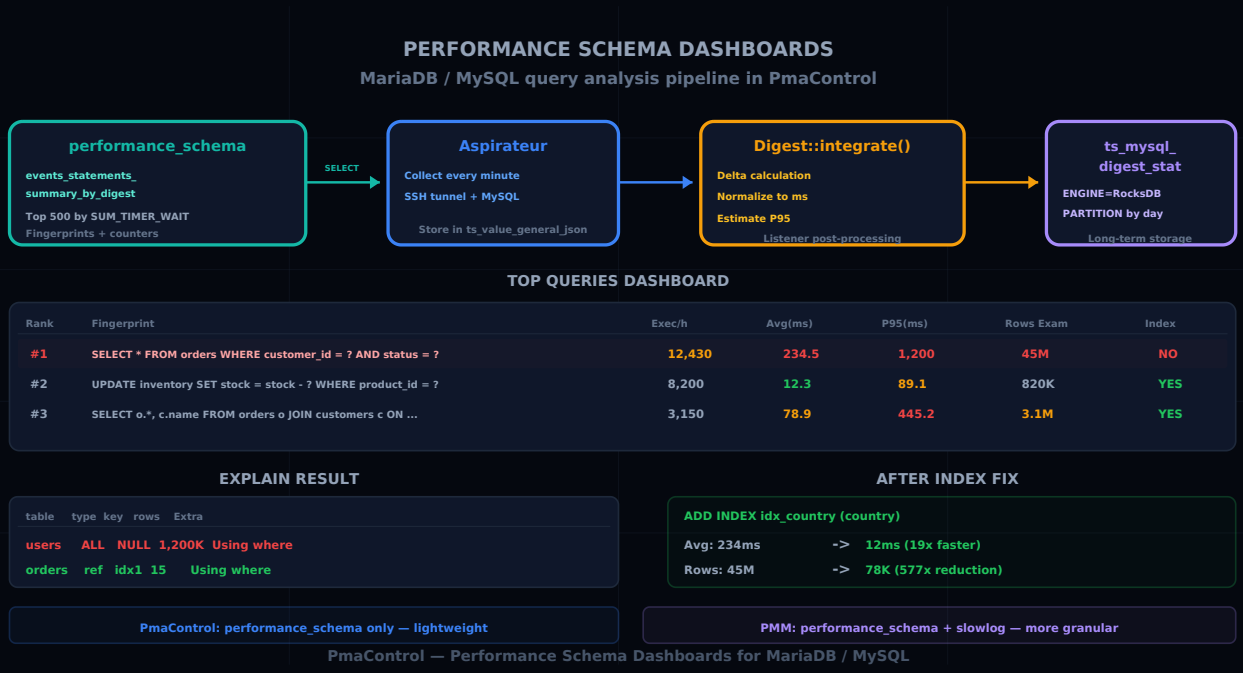
Performance Schema et dashboards PmaControl : traquer les requêtes lentes

Aurélien LEQUOY · 13 avril 2026

MARIADB MYSQL PERFORMANCE-SCHEMA DIAGNOSTICS PMACONTROL



Performance Schema : la mine d'or inexploitée

`performance_schema` est activé par défaut dans MariaDB / MySQL depuis des années. Et pourtant, la majorité des DBA ne l'exploitent pas au quotidien. La raison est simple : les données brutes sont difficilement lisibles. Des dizaines de tables, des millions de lignes, des compteurs cumulatifs — sans outil d'agrégation, c'est du bruit.

PmaControl transforme ce bruit en signal. Il collecte les données de `performance_schema` via l'Aspirateur, les agrège via le Listener (Digest::integrate), et les présente dans des dashboards exploitables. Cet article explique le pipeline complet, de la source au dashboard.

Vérifier que performance_schema est activé

MariaDB

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

Si `OFF`, ajoutez dans le fichier de configuration :

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

Un redémarrage est nécessaire — cette variable n'est pas dynamique.

MySQL

Même chose sur MySQL. La variable est en lecture seule et nécessite un redémarrage :

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

Impact sur les performances

La question classique : "est-ce que performance_schema ralentit mon serveur ?" La réponse en 2026 est **non, de façon mesurable**. L'overhead est de l'ordre de 1-3% sur les workloads typiques. Le bénéfice en visibilité compense largement ce coût.

La seule exception : les serveurs avec des workloads extrêmes (>100 000 requêtes/seconde) où chaque pourcentage compte. Dans ce cas, désactivez les instruments non nécessaires plutôt que performance_schema entier.

La source : events_statements_summary_by_digest

La table clé que PmaControl exploite est :

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

Cette table contient un résumé par **fingerprint** (empreinte normalisée) de chaque requête exécutée. Voici les colonnes les plus utiles :

Colonne	Description
`DIGEST`	Hash unique du fingerprint
`DIGEST_TEXT`	Texte normalisé de la requête (paramètres remplacés par `?`)
`COUNT_STAR`	Nombre total d'exécutions
`SUM_TIMER_WAIT`	Temps total d'exécution (en picosecondes)
`AVG_TIMER_WAIT`	Temps moyen par exécution
`SUM_ROWS_EXAMINED`	Total de lignes examinées
`SUM_ROWS_SENT`	Total de lignes retournées
`FIRST_SEEN`	Première exécution
`LAST_SEEN`	Dernière exécution

Le fingerprint est la clé de voûte : il normalise `SELECT * FROM users WHERE id = 42` et `SELECT * FROM users WHERE id = 1337` en un seul fingerprint `SELECT * FROM users WHERE id = ?`. Cela permet d'agréger les statistiques indépendamment des valeurs de paramètres.

Le pipeline PmaControl

Étape 1 : Collecte par l'Aspirateur

L'Aspirateur exécute périodiquement la requête suivante sur chaque serveur supervisé :

```
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;
```

Le `LIMIT 500` est intentionnel : PmaControl se concentre sur les 500 requêtes les plus coûteuses en temps cumulé. Les requêtes rapides et peu fréquentes ne sont pas intéressantes pour l'optimisation.

Les résultats sont stockés dans `ts_value_general_json` avec un horodatage.

Étape 2 : Traitement par le Listener

Quand le Listener détecte de nouvelles données de digest, il déclenche `Digest::integrate()`. Cette fonction :

1. **Calcule les deltas** : puisque `performance_schema` fournit des compteurs cumulatifs (depuis le dernier `TRUNCATE` ou redémarrage), Digest::integrate calcule la différence entre deux collectes pour obtenir les métriques de la période.
2. **Normalise les temps** : les picosecondes sont converties en millisecondes pour l'affichage.
3. **Calcule les percentiles** : le P95 (95e percentile) du temps d'exécution est estimé à partir des distributions.
4. **Stocke dans ts_mysql_digest_stat** : la table dédiée aux statistiques de digest, partitionnée par jour et utilisant le moteur RocksDB pour la compression.

```
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
```

La table ts_mysql_digest_stat

C'est le stockage long terme des statistiques de 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')),
    ...
);
```

Le partitionnement par jour permet :

- Un nettoyage rapide : `ALTER TABLE ts_mysql_digest_stat DROP PARTITION p20260401;`
- Des requêtes rapides sur une plage de dates
- Un contrôle fin de la rétention

Les dashboards

Vue Top Queries

Le dashboard principal affiche les requêtes les plus coûteuses, triées par temps cumulé :

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

Chaque ligne est cliquable pour accéder au détail.

Vue détaillée d'un fingerprint

En cliquant sur un fingerprint, PmaControl affiche :

- **Le texte complet** de la requête normalisée
- **L'historique** : évolution du temps d'exécution moyen et du P95 sur les 30 derniers jours
- **Le ratio** rows_examined / rows_sent — un ratio élevé (>100:1) indique un scan de table ou un index manquant
- **Le flag no_index_used** — combien d'exécutions n'ont utilisé aucun index

Identifier les index manquants

Le ratio rows_examined / rows_sent est l'indicateur le plus puissant. Prenons un exemple :

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

Ce ratio de 100:1 signifie que MariaDB / MySQL examine 100 lignes pour en retourner 1. C'est le signe classique d'un full table scan ou d'un index inefficace.

Action : vérifier la présence d'un index sur `customer_id` :

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

Si l'index n'existe pas :

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

Le flag SUM_NO_INDEX_USED

PmaControl affiche en rouge les requêtes où `SUM_NO_INDEX_USED` est élevé. Ce flag est activé quand MariaDB / MySQL exécute un full table scan — c'est souvent le problème de performance numéro un.

EXPLAIN depuis PmaControl

Pour les requêtes identifiées comme problématiques, PmaControl peut exécuter un `EXPLAIN` directement :

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

Le résultat est affiché avec un code couleur :

- **Vert** : `type = ref` ou `type = eq_ref` — utilisation d'index, bon
- **Ambre** : `type = range` — scan de plage, acceptable
- **Rouge** : `type = ALL` — full table scan, à corriger

Intégration avec Marina+ agent

Marina+ est l'agent d'optimisation automatique de PmaControl. Il analyse les données de digest et propose des suggestions :

1. **Index manquants** : détecte les requêtes avec un ratio rows_examined/rows_sent élevé et suggère les index à créer
2. **Requêtes à réécrire** : identifie les patterns inefficaces (SELECT *, subqueries corrélées, ORDER BY sur colonne non indexée)
3. **Configuration** : ajuste les paramètres serveur basés sur les patterns de requêtes (sort_buffer_size, join_buffer_size, etc.)

Marina+ ne modifie rien automatiquement — il génère des recommandations que le DBA valide et applique.

Comparaison avec PMM (Percona Monitoring and Management)

PMM et PmaControl exploitent la même source de données (`performance_schema`), mais avec des approches différentes :

Aspect	PmaControl	PMM
Source	performance_schema	performance_schema + slowlog
Agent	Aspirateur (SSH + MySQL)	mysqld_exporter + QAN
Stockage	ts_mysql_digest_stat (RocksDB)	ClickHouse (QAN)
Fingerprinting	Serveur-side (MariaDB / MySQL natif)	Client-side (Percona agent)
P95	Estimé à partir des distributions	Calculé à partir du slowlog
Historique	Partitionné par jour, rétention configurable	ClickHouse, rétention configurable
Actions	EXPLAIN intégré, suggestions Marina+	Query Analytics + PMM UI

La différence principale : **PMM combine performance_schema et slowlog** pour des percentiles plus précis. PmaControl se base uniquement sur performance_schema, ce qui est plus léger mais moins granulaire.

L'avantage de PmaControl : l'intégration avec le reste de l'écosystème (réplication, topologie, alertes, actions). PMM est meilleur pour l'analyse pure de requêtes, PmaControl est meilleur pour la vision globale de l'infrastructure.

Cas pratique : trouver et corriger une requête lente

Scénario : le dashboard PmaControl montre une requête qui consomme 40% du temps total du serveur.

Étape 1 : Identifier

Dans le dashboard 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
```

Étape 2 : Analyser

Le ratio rows_examined / rows_sent est catastrophique : 45 millions de lignes examinées pour ~5 200 résultats par heure.

EXPLAIN depuis PmaControl :

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

Le problème : `users` est scanné en full table (`type = ALL`). Il n'y a pas d'index sur `country` .

Étape 3 : Corriger

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

Étape 4 : Vérifier

Après l'ajout de l'index, le dashboard PmaControl montre l'amélioration dans l'heure :

```
#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
```

Le temps moyen est passé de 234ms à 12ms (x19), et les lignes examinées de 45M à 78K (x577).

Bonnes pratiques

1. Ne pas TRUNCATE performance_schema manuellement

PmaControl calcule des deltas entre deux collectes. Si vous faites `TRUNCATE TABLE performance_schema.events_statements_summary_by_digest`, les compteurs redémarrent à zéro et le premier delta sera incorrect. Laissez PmaControl gérer.

2. Augmenter performance_schema_digests_size si nécessaire

Par défaut, MariaDB / MySQL stocke les N premiers fingerprints. Si votre application a plus de requêtes distinctes que la limite, les moins fréquentes sont évincées :

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

3. Corréler avec le slow query log

PmaControl via performance_schema donne le "quoi" (quelles requêtes sont lentes). Le slow query log donne le "quand" (à quel moment exact). Les deux sont complémentaires.

4. Surveiller le ratio rows_examined / rows_sent

C'est l'indicateur le plus actionnable. Un ratio > 100:1 est presque toujours un index manquant. Un ratio > 1000:1 est un problème urgent.

5. Utiliser le P95, pas la moyenne

La moyenne masque les outliers. Une requête avec une moyenne de 10ms mais un P95 de 500ms a un problème intermittent (lock contention, cold cache, plan d'exécution instable). Le P95 révèle ces problèmes.

Conclusion

Performance_schema est la meilleure source de données pour l'optimisation des requêtes MariaDB / MySQL. PmaControl automatise la collecte, l'agrégation et la présentation de ces données via le pipeline Aspirateur → Digest::integrate → ts_mysql_digest_stat → dashboard.

Le résultat : une visibilité continue sur les requêtes les plus coûteuses, les index manquants, et l'évolution des performances dans le temps. Combiné avec Marina+ pour les suggestions automatisées, c'est un workflow complet d'optimisation des performances — de la détection à la correction.