Cache et index présents, page lente : une enquête MariaDB dans PmaControl

Aurélien LEQUOY · 9 octobre 2026

PMACONTROL MARIADB PERFORMANCE SQL BENCHMARK ARCHITECTURE



« Les dessins sont déjà en cache. Pourquoi la page Architecture met-elle encore plusieurs secondes à s'afficher ? »

Cette question nous a conduits à une correction très courte dans PmaControl. **Il ne manquait pas d'index. Il n'était pas non plus nécessaire de recalculer les dessins.** MariaDB avait choisi une manière coûteuse de retrouver la génération de graphes à afficher.

Voici les mesures, le SQL et les limites de cette comparaison. Nous distinguons **un client** qui présentait le problème et une instance de test qui exécutait déjà rapidement la requête initiale.

Mise à jour du 10 octobre 2026. Le relevé SQL de 13 h 29 est ajouté ci-dessous. Les temps de téléchargement ont été remesurés vers 13 h 45 depuis un serveur sur le réseau local, car la connexion utilisée pour les captures du 9 octobre était instable. Chaque campagne conserve son environnement, son volume de graphes et son périmètre : SQL, téléchargement HTTP ou

chargement dans Chromium.

L’explication avec une boîte de dessins

Imaginez une bibliothèque remplie de boîtes. Chaque boîte contient les dessins d’une génération de la topologie. Les dessins sont déjà terminés : c’est notre cache de SVG.

Pour afficher la page, il reste à retrouver la bonne boîte, à prendre ses dessins, puis à les disposer sur l’écran. Le catalogue de la bibliothèque existe : c’est l’index SQL.

Mais avoir un catalogue ne suffit pas. Dans le cas lent, MariaDB choisissait de parcourir des millions d’entrées pour préparer une liste intermédiaire de correspondances. Pour la vérification qui nous intéresse, trouver **une seule correspondance** suffit.

La correction rend cette vérification explicite. Les mêmes dessins sont renvoyés ; la recherche évite le détour coûteux.

Ce qui était déjà en cache

Les SVG sont enregistrés dans `dot3_graph`. La table `dot3_cluster` les relie aux générations de topologie. La page récupère déjà tous les graphes avec **une seule requête SQL**, qui renvoie notamment :

```
SELECT dg.id, dg.svg, dg.height, dg.width,
       (dg.height * dg.width) AS area,
       dc.date_inserted AS date_refresh
FROM dot3_cluster dc
JOIN dot3_graph dg ON dg.id = dc.id_dot3_graph
-- La condition choisit la génération prête à afficher.
```

Le navigateur reçoit donc le dessin, sa hauteur, sa largeur et sa surface. Le code de `App/Webroot/js/Architecture/topology.js` réalise déjà le placement dans le navigateur. Cette correction ne modifie pas cet algorithme.

Le cache évite de fabriquer les dessins à chaque visite. Il n’évite pas de rechercher les dessins, de les transmettre et de les afficher. Ce cache de SVG est distinct du cache de requêtes de MariaDB et du cache HTTP du navigateur.

Non, il ne manquait pas d’index

L’instance lente possédait déjà cet index unique, tel que confirmé par `SHOW INDEX FROM dot3_cluster` :

```
-- Index existant, présenté sous forme de définition :
UNIQUE KEY id_dot3_information_2
    (id_dot3_information, id_dot3_graph)
```

La recherche filtre sur `id_dot3_information`, la première colonne de l’index. Celui-ci permet donc de retrouver les clusters d’une génération.

Le problème se trouvait dans le **plan d’exécution** : la stratégie choisie par MariaDB pour exécuter le SQL. La version initiale utilisait `EXISTS` pour vérifier qu’une génération possède au moins un cluster. Sur cette instance, ce choix entraînait une matérialisation : MariaDB préparait un résultat intermédiaire en parcourant environ **4,33 millions d’entrées historiques**.

Cela ne signifie pas que le navigateur recevait quatre millions de graphes. Il recevait **61 graphes** . Le travail excessif se produisait dans la base, avant la récupération du résultat utile. Le chiffre de 4,33 millions décrit l’ordre de grandeur du parcours identifié pendant le premier diagnostic, pas un comptage de lignes transférées à PHP. Le relevé du 10 octobre indique désormais environ **5,2 millions de lignes** dans la table de cette instance ; les volumes de l’article initial et de la nouvelle campagne ne sont donc pas strictement identiques.

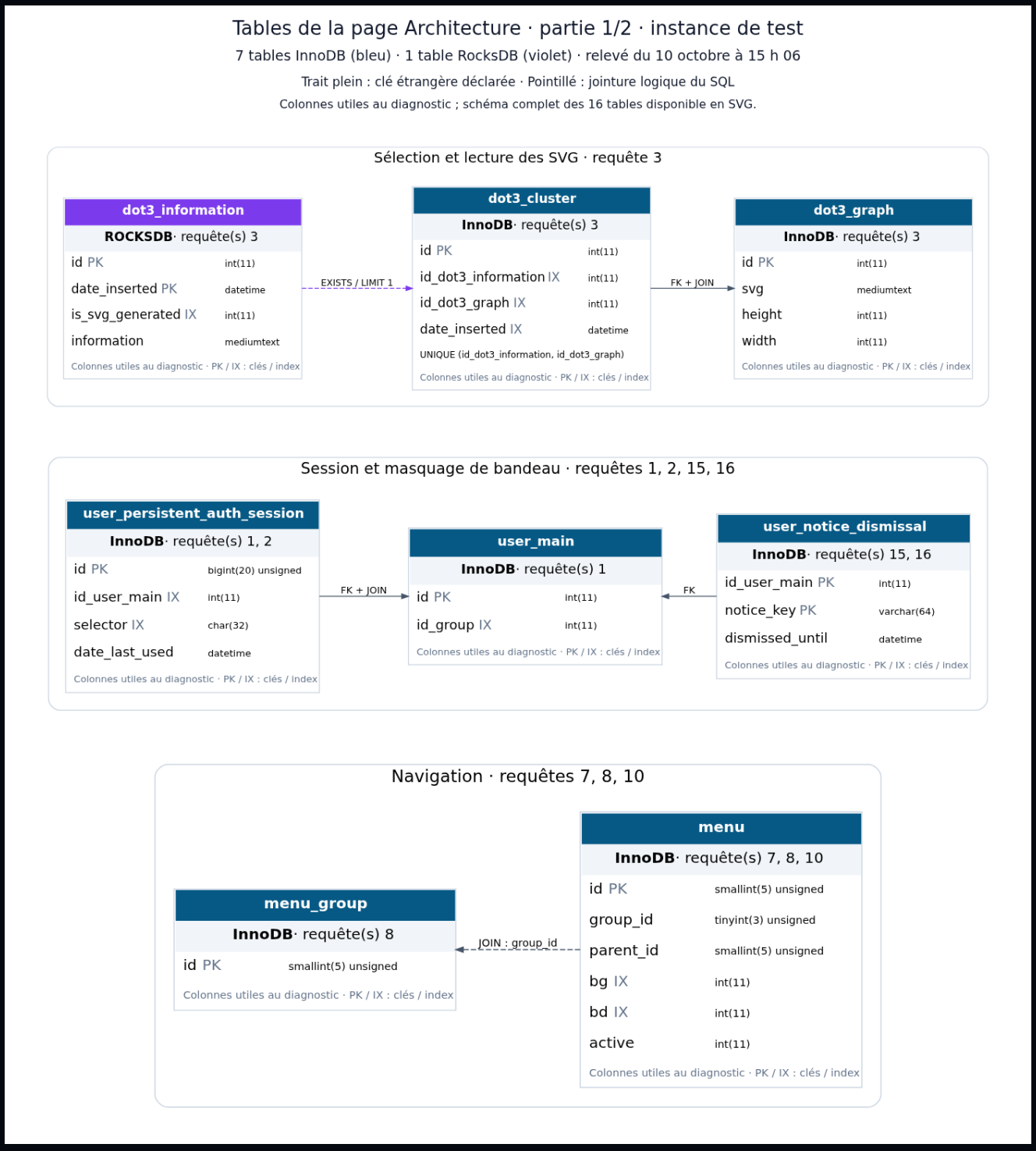
Un index peut être utilisé lors d’un parcours très large. La présence d’un index ne garantit donc pas, à elle seule, un accès court.

MariaDB, configuration et tables InnoDB / RocksDB

Les deux environnements utilisent **MariaDB 11.8.9**, avec des paquets différents : **11.8.9-deb13 chez un client**, d’après son relevé du 10 octobre, et **11.8.9-MariaDB-deb12-log sur l’instance de test**, vérifié directement sur le serveur. Le client utilise PHP 8.4 pour le rejeu SQL ; l’instance de test utilise FrankenPHP / PHP 8.5.9.

Le schéma ci-dessous couvre **les 16 tables applicatives utilisées par les 16 requêtes de la page**, au-delà des trois tables de la requête SVG. Les colonnes utiles, les clés et les moteurs proviennent du schéma réel de l’instance de test, relevé le **10 octobre à 15 h 06**. Les vérifications de présence des tables interrogent aussi `information_schema.tables`, une vue de

métadonnées qui n’est pas une table applicative à classer en InnoDB ou RocksDB.





Ouvrir le schéma vectoriel en grand. Un trait plein indique une clé étrangère déclarée ; un pointillé indique une jointure logique utilisée par le SQL. Les tables sans liaison affichée sont interrogées séparément. Le dessin présente les relations, pas l’ordre d’exécution des requêtes.

Table de la requête SVG sur le test	Moteur constaté
`dot3_information`	**RocksDB**
`dot3_cluster`	**InnoDB**
`dot3_graph`	**InnoDB**

Les treize autres tables de la page sont en InnoDB. Le schéma complet ci-dessus et l'inventaire téléchargeable des moteurs et tailles les détaillent.

La requête SVG traverse donc **deux moteurs** sur l'instance de test. L'index de disponibilité de `dot3_information` appartient à RocksDB ; l'index unique de `dot3_cluster` et les SVG de `dot3_graph` appartiennent à InnoDB. Le correctif garde ces moteurs et ces index. Cette campagne compare deux formulations SQL ; elle ne mesure pas une conversion de tables d'InnoDB vers RocksDB. Le relevé du client ne fournit pas ses moteurs : nous ne lui attribuons pas le schéma observé sur le test.

La configuration `.cnf` de l'instance de test, anonymisée, réunit les sections serveur des fichiers inclus dans leur ordre de lecture. Les secrets et identifiants réseau sont retirés. Les fichiers comportent des réglages successifs : les variables globales effectivement relevées permettent de voir les valeurs actives, notamment **2 Gio de buffer pool InnoDB**, **2 Gio de cache de blocs RocksDB**, **128 connexions maximum**, **64 Mio pour `tmp_table_size` et `max_heap_table_size`**, et le **cache de requêtes désactivé**.

Ces paramètres ont été collectés à **15 h 14**, après les mesures de 13 h 45 ; certaines variables peuvent changer dynamiquement. La ressource `.cnf` décrit la configuration observée et ses surcharges, avec des valeurs retirées ; elle n'est pas un fichier à appliquer directement. La configuration `.cnf` du client n'est pas disponible dans le relevé.

La modification SQL, avant et après

Le changement se situe dans `App/Library/Dot3LatestGeneration.php`, au sein de la sélection de la dernière génération prête.

Avant :

```
AND EXISTS (
    SELECT 1
    FROM dot3_cluster dc_ready
    WHERE dc_ready.id_dot3_information = di.is_svg_generated
)
```

Après :

```
AND (
    SELECT dc_ready.id_dot3_information
    FROM dot3_cluster dc_ready
    WHERE dc_ready.id_dot3_information = di.is_svg_generated
    LIMIT 1
) IS NOT NULL
```

Pourquoi est-ce équivalent ici ? La requête extérieure impose déjà `di.is_svg_generated > 0`. S'il existe un cluster correspondant, la sous-requête renvoie cet identifiant positif. Sinon, une sous-requête scalaire sans résultat renvoie `NULL`. La condition vérifie donc toujours l'existence d'un cluster pour cette génération.

La sous-requête scalaire évite le plan de matérialisation observé avec `EXISTS`. Le moteur peut rechercher la génération dans l'index et s'arrêter à la première correspondance. Les solutions de repli utilisées si aucune génération prête n'est disponible restent identiques.

Ce n'est pas une règle disant que `EXISTS` est lent. Sur notre autre instance, l'ancienne requête utilisait déjà une stratégie `FirstMatch` efficace. Réécrire tous les `EXISTS` serait une mauvaise conclusion : il faut examiner et mesurer le plan de chaque requête.

Un client : nouveau relevé SQL du 10 octobre

Le relevé réalisé chez **un client** vers **13 h 29** compare dix passages par variante sur les données de production, après un passage de chauffe exclu, en alternant l'ordre avant/après. Le code de production contient déjà la correction : « avant » désigne ici l'exécution de l'ancienne requête `EXISTS` pour comparaison, et « après » la sous-requête scalaire avec `LIMIT 1`.

Mesure SQL	Avant (`EXISTS`)	Après (`LIMIT 1`)
Requête SVG, minimum–maximum	4 434,17–5 297,38 ms	2,69–6,20 ms
Requête SVG, médiane	**5 044,59 ms**	**3,32 ms**
Total des 16 requêtes, médiane	5 068,52 ms	12,92 ms
Graphes renvoyés	61	61
Volume des SVG	995 307 octets	995 307 octets
Compteur de lectures d'index du relevé	5 199 176	3

La requête dominante passe de **5,04 secondes à 3,32 ms**, soit environ **1 520 fois plus rapide** dans ce relevé. Les 61 graphes sont identiques avant et après la correction. Le volume actuel diffère de 130 octets du premier diagnostic, qui relevait 995 437 octets ; ce sont deux dates de mesure, tandis que les deux variantes de la nouvelle campagne renvoient le même contenu.

Le plan `EXPLAIN` avant **estime 4 406 092 entrées** pour la matérialisation ; après, il **estime 81 entrées** pour la sous-requête dépendante utilisant l'index. Ces estimations ne sont pas des comptages de lectures réellement effectuées. Le compteur de lectures du relevé indique **5 199 176 avant et 3 après** ; `dot3_cluster` contient alors **5 198 990 lignes**. Estimation du plan, lectures relevées et graphes transmis sont trois mesures distinctes.

Les **médianes des 15 autres requêtes restent toutes sous 10 ms**. Certaines fluctuent : le comptage des alarmes passe par exemple de 9,21 à 0,85 ms et atteint ponctuellement 15,55 ms avant. Ce résultat ne démontre pas que le correctif accélère le comptage : le serveur avait une charge d'environ 6 et ces requêtes suivaient immédiatement une requête de cinq secondes. Le changement de code ne concerne que la sélection des SVG.

La seule requête de modification rejouée par cette campagne, l'actualisation de la session, a été exécutée puis annulée. **Ce relevé mesure le SQL ; il ne mesure ni le téléchargement ni le chargement complet du navigateur.** Les temps incluent la récupération complète des résultats par mysqli, dont environ 1 Mo de SVG.

Requête chez un client	Avant (ms)	Après (ms)
1. Lire la session de connexion	0,34	0,34
2. Actualiser la session, puis annuler	0,16	0,19
3. Charger les 61 graphes et leurs dimensions	5 044,59	3,32
4. Lister les clients	0,53	0,29
5. Lister les environnements	0,78	0,15
6. Lister les tags	0,60	0,22
7. Chercher le fil de navigation	1,59	0,64
8. Construire le menu	5,26	4,24
9. Compter les alarmes actives	9,21	0,85

Requête chez un client	Avant (ms)	Après (ms)
10. Sélectionner la rubrique du menu	0,78	0,66
11. Vérifier la clé de traduction	0,53	0,15
12. Lire la traduction	0,48	0,18
13. Vérifier la table de configuration	0,20	0,17
14. Lire les paramètres de licence	0,17	0,15
15. Vérifier la table des bandeaux masqués	0,18	0,15
16. Lire le masquage du bandeau de licence	0,30	0,13
Total, médiane des sommes par passage	**5 068,52**	**12,92**

Le relevé SQL anonymisé avec les plages de temps et les deux plans complète ce tableau.

Téléchargement chez un client : hypothèse de comparaison

Pour compléter la comparaison HTTP, nous **reprenons les temps de téléchargement mesurés sur l'instance de test depuis le serveur**, présentés dans la section suivante. Nous supposons ces durées identiques chez un client. **Ces valeurs sont une hypothèse commune aux deux environnements ; elles ne constituent pas une nouvelle mesure HTTP chez le client.**

Téléchargement du document HTML, repris par hypothèse	Avant	Après
Médiane Chromium de l'instance de test	42,90 ms	37,45 ms

Le premier octet et le chargement complet chez le client n'ont pas été remesurés. Ils ne peuvent pas reprendre les totaux du test : avant, la requête SVG du client prend à elle seule cinq secondes. Nous gardons séparés le SQL mesuré et le transfert supposé, sans fabriquer un temps navigateur.

Téléchargement et navigateur : nouvelle mesure depuis un serveur

Pour éliminer l’incertitude liée à la connexion des anciennes captures, une autre campagne a été réalisée le **10 octobre vers 13 h 45**, depuis un serveur Debian 12 vers l’instance de test sur le réseau local. Elle porte sur la même page Architecture sans filtre, le même compte temporaire de lecture seule, **30 graphes**, un écran de 1 440 × 900 pixels et Chromium 152. Le cache HTTP du navigateur est désactivé ; un chargement de chauffe par variante est exclu, puis **dix chargements avant et dix après** sont conservés.

Le serveur cible utilise MariaDB 11.8.9 et FrankenPHP / PHP 8.5.9. Le code applicatif est celui du commit `4207dfbe5e7d1709dbbd2efdd2cff8c1e6619dc1` ; pour la variante avant, seul `Dot3LatestGeneration.php` est repris du parent `366b200703ff7550ec01d33e2c7d49c0b570ab15`. Les mesures passent par un serveur HTTPS temporaire du worktree sur le réseau local. La production reste sur le code corrigé.

Médianes en millisecondes :

Étape mesurée dans Chromium	Avant (`EXISTS`)	Après (`LIMIT 1`)
Premier octet, PHP et transport inclus	62,35	50,90
Téléchargement du document HTML	42,90	37,45
Document reçu → événement `load`	416,05	392,20
Chargement complet → `load`	**523,15**	**488,40**



Le téléchargement du document varie de **33,2 à 61,3 ms avant**, et de **32,3 à 55,1 ms après**. Le chargement complet varie de **472,3 à 681,5 ms avant**, et de **444,5 à 669,7 ms après**. Ces séries se recouvrent : elles montrent un chargement autour d'une demi-seconde sur ce trajet, sans établir un gain navigateur reproductible de 34,75 ms.

Les **30 SVG affichés ont le même SHA-256 à chacun des vingt chargements**. Le document décompressé mesure **2 240 909 octets** dans les deux variantes. Son corps compressé représente environ **671 254 octets**, soit environ 671 554 octets avec les en-têtes comptés par Chromium. Ce volume HTML et ces 30 graphes appartiennent à l'instance de test ; ils ne correspondent pas aux 995 307 octets de SVG des 61 graphes chez un client.

Un contrôle distinct avec `curl --compressed`, dix requêtes par variante après chauffe, confirme que le transfert reste variable : téléchargement médian de **70,85 ms avant et 93,80 ms après**, réponse HTTP complète de **119,97 et 145,62 ms**. `curl` ne charge pas les ressources de la page et n'exécute pas le JavaScript. La décompression, le lecteur HTTP, le protocole négocié et la charge du serveur contribuent aux écarts avec Chromium ; ces écarts ne prouvent pas une régression du SQL.

Les étapes sont mesurées pour chaque visite, puis résumées séparément : **la somme des médianes n'est pas nécessairement la médiane du total**. Le temps après réception du document inclut d'autres ressources, l'analyse HTML, du JavaScript et du rendu. Nous ne

l’attribuons pas au seul placement des graphes. Le premier octet inclut déjà la préparation PHP ; aucun temps PHP estimé n’est ajouté au total.

Les anciens chargements de 12 948, 2 952 et 843 ms restent des observations historiques du 9 octobre, disponibles dans le CSV d’archive. Leur téléchargement de 470 à 535 ms dépendait de la connexion alors instable : ils ne servent plus de référence pour le réseau. La nouvelle campagne sur 30 graphes ne reconstitue pas le temps navigateur de l’instance à 61 graphes.

La nouvelle comparaison SQL sur l’instance de test, dix paires alternées dans un même instantané en lecture seule avec le cache de requêtes MariaDB désactivé pour la session, donne **6,94 ms avant et 5,81 ms après**. Les résultats complets et les **2 172 396 octets de SVG** sont identiques. Le plan initial y utilise toujours `FirstMatch` : même avec 6 591 509 lignes dans `dot3_cluster`, cette instance ne reproduit pas la matérialisation coûteuse de l’autre serveur. La taille d’une table seule ne suffit donc pas à prédire la lenteur.

Instance de test : détail des 16 requêtes du 9 octobre

Le tableau détaillé ci-dessous conserve la campagne instrumentée du **9 octobre**, sur la page sans filtre, avec un compte de lecture seule et **30 graphes**. Un premier chargement de chauffe est exclu, puis cinq chargements sont conservés pour chaque version. Ce tableau n’est pas présenté comme une nouvelle mesure des 16 requêtes du 10 octobre.

Le profileur de l’application relève **16 requêtes à chacun des dix chargements**. Chaque ligne ci-dessous correspond à une requête exécutée une fois avant et une fois après ; la dernière ligne indique le total. Les temps ci-dessous sont les médianes des appels SQL mesurés côté pilote PHP, en millisecondes. Ils ne représentent pas uniquement le temps interne du moteur MariaDB.

Requête	Avant (ms)	Après (ms)
1. Lire la session de connexion	0,31	0,34
2. Actualiser la session	0,29	0,27
3. Charger les 30 graphes et leurs dimensions	5,86	5,57
4. Lister les clients	0,37	0,34
5. Lister les environnements	0,23	0,27

Requête	Avant (ms)	Après (ms)
6. Lister les tags	0,23	0,29
7. Chercher le fil de navigation	0,71	1,07
8. Construire le menu	3,02	3,04
9. Compter les alarmes actives	1,11	1,09
10. Sélectionner la rubrique du menu	0,67	0,67
11. Vérifier la clé de traduction	0,33	0,31
12. Lire la traduction	0,30	0,29
13. Vérifier la table de configuration	0,33	0,31
14. Lire les paramètres de licence	0,22	0,23
15. Vérifier la table des bandeaux masqués	0,32	0,31
16. Lire le masquage du bandeau de licence	0,18	0,22
Total, médiane	**14,53**	**14,59**

Le total est la médiane des sommes calculées pour chaque chargement. Additionner les médianes de chaque ligne ne donne pas nécessairement cette valeur.

Sur cette instance de test, **aucun gain SQL global significatif n'est démontré**. La différence de 0,06 ms est minime. Cela s'explique : MariaDB y choisissait déjà un bon plan pour l'ancienne requête. Le correctif vise le plan coûteux rencontré sur l'instance de production.

Ce comptage couvre les requêtes applicatives du rendu PHP initial, sur les trois connexions relevées par le profileur. Il exclut les appels AJAX ultérieurs et les commandes internes de connexion. Il ne faut pas attribuer ces 16 requêtes à tous les profils utilisateurs ou à toutes les installations.

Vérifier aussi que les résultats restent corrects

Une page plus rapide n'est utile que si elle affiche les bons graphes. La vérification a porté sur l'égalité des résultats avant/après, les générations prêtes ou absentes et les solutions de repli.

Neuf scénarios ont été exécutés avec des données de test définies dans des CTE MariaDB, sans insertion dans les tables applicatives. Des comparaisons SQL ont également porté sur la page sans filtre, la page filtrée et l'agrégation de graphes, avec les résultats comparés dans le même instantané transactionnel. Le test JavaScript de placement a vérifié le comportement sur plusieurs largeurs d'écran.

Une série distincte de quinze mesures appariées sur l'instance de test a relevé de petits surcoûts médians sur les chemins filtré et agrégé : environ **0,34 ms** et **0,16 ms**. Nous les conservons dans le bilan : la correction ne rend pas tous les scénarios plus rapides. Ces contrôles ciblés ne remplacent pas l'ensemble de la suite de tests du produit.

Refaire le diagnostic sur une autre page

Commencez par séparer trois questions : combien de temps le serveur prépare-t-il la réponse, combien de temps le document met-il à arriver et combien de temps le navigateur met-il à terminer le chargement ?

Ensuite, comptez les requêtes et mesurez chacune d'elles. Pour celle qui domine, comparez le plan avant/après et l'égalité des résultats. Vérifiez notamment le nombre d'entrées parcourues, les résultats intermédiaires et l'utilisation effective des index. Répétez les mesures : une seule observation peut être trompeuse.

Dans ce cas, **une requête est restée une requête**, et l'index est resté le même. C'est la façon de retrouver les dessins déjà en cache qui a changé.

Données et SQL détaillés

- Les 16 requêtes, leur emplacement dans le code et le SQL complet avant/après
- Les dix chargements instrumentés : durée de chaque requête en CSV
- Les vingt chargements Chromium du 10 octobre depuis le serveur, en CSV
- Le contrôle HTTP curl du 10 octobre, avant et après
- La comparaison SQL du 10 octobre sur l'instance de test : résultats et plans en JSON
- Le relevé SQL anonymisé chez un client : 16 requêtes, plages et plans
- La configuration MariaDB `.cnf` anonymisée de l'instance de test et les variables actives, la version et les fichiers inclus

- Le schéma des 16 tables en SVG, les définitions SQL, l’inventaire des moteurs et tailles en CSV et les colonnes, index et relations en JSON
- Le script SQL, fourni en texte, le script Chromium, le script HTTP et la configuration du serveur HTTPS temporaire
- Méthode et étapes de reproduction de la campagne du 10 octobre
- Archive : les trois chargements de production du 9 octobre sur la connexion instable

Les identifiants de session et de compte sont remplacés par des paramètres dans les exemples téléchargeables. Les mesures de production et de test sont séparées pour permettre de les interpréter sans mélanger les deux environnements.