

Optimisation des performances

Introduction

Lorsqu'une base de données grandit, les temps de réponse peuvent se dégrader significativement. Une requête qui s'exécute en quelques millisecondes sur 1 000 lignes peut prendre plusieurs secondes sur 5 millions de lignes.

L'optimisation des performances repose sur **trois leviers** :

- **Les index** : accélérer les recherches
- **La configuration serveur** : adapter les ressources
- **Les requêtes** : écrire des requêtes efficaces

Les index

Principe

Un **index** fonctionne comme l'index d'un livre : plutôt que de parcourir toutes les pages pour trouver un mot, on consulte l'index qui indique directement la bonne page.

Sans index, le SGBD effectue un **full table scan** : il parcourt *toutes* les lignes de la table pour trouver celles qui correspondent à la condition.

Les index ont un **coût** : ils occupent de l'espace disque et ralentissent les opérations d'écriture (INSERT, UPDATE, DELETE) car l'index doit être mis à jour à chaque modification.

Index simple

Un index simple porte sur **une seule colonne**.

```
-- Créer un index sur la colonne planete_origine
CREATE INDEX idx_planete ON transmissions(planete_origine);

-- Supprimer un index
DROP INDEX idx_planete ON transmissions;
```

Créez un index simple sur les colonnes fréquemment utilisées dans :

- Les clauses `WHERE`
- Les clauses `JOIN`
- Les clauses `ORDER BY`

Index composé

Un index composé porte sur **plusieurs colonnes**. L'ordre des colonnes est crucial.

```
-- Index sur deux colonnes
CREATE INDEX idx_planete_priorite ON transmissions(planete_origine, priorite);
```

Placez en premier dans l'index les colonnes les plus **sélectives** (celles qui filtrent le plus de lignes).

Index unique

Un index unique garantit que les valeurs de la colonne sont **uniques** dans toute la table, tout en offrant les mêmes gains de performance qu'un index classique.

```
-- Garantir l'unicité du code de transmission
CREATE UNIQUE INDEX idx_code_unique ON transmissions(code_transmission);
```

Différence avec **PRIMARY KEY** :

- `PRIMARY KEY` : une seule par table, n'accepte pas les NULL
- `UNIQUE INDEX` : plusieurs possibles par table, accepte les NULL

Analyser une requête avec EXPLAIN

La commande `EXPLAIN` permet de voir comment MariaDB exécute une requête.

```
EXPLAIN SELECT * FROM transmissions WHERE planete_origine = 'Tatooine';
```

Résultat simplifié :

type	possible_keys	key	rows
ALL	NULL	NULL	5000000

Les colonnes importantes :

- **type** : le type de scan
 - `ALL` = full table scan (mauvais)
 - `ref` = utilisation d'un index (bon)
 - `range` = scan partiel d'index (bon)
 - `const` = résultat unique via PRIMARY KEY (excellent)
- **key** : l'index utilisé (NULL = aucun index)
- **rows** : estimation du nombre de lignes parcourues

Si vous voyez `type = ALL` sur une grande table, c'est un signal d'alarme : un index est probablement nécessaire.

Exemple pratique : Registre galactique des communications

Mettons en pratique ces concepts avec une table de **5 millions de lignes**.

Contexte

L'Empire intercepte et enregistre toutes les communications de la galaxie. Chaque transmission est stockée avec sa date, son origine, sa priorité et son contenu.

Création de la table

```
CREATE TABLE transmissions (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  code_transmission VARCHAR(20) NOT NULL,  
  date_emission DATETIME NOT NULL,  
  emetteur VARCHAR(100) NOT NULL,  
  recepteur VARCHAR(100) NOT NULL,  
  planete_origine VARCHAR(50) NOT NULL,  
  priorite ENUM('basse', 'normale', 'haute', 'urgente') NOT NULL,  
  contenu TEXT
```

```
) ENGINE=InnoDB;
```

Génération de 5 millions de lignes

Cette procédure génère des données aléatoires :

```
DELIMITER //

CREATE PROCEDURE generer_transmissions(IN nb_lignes INT)
BEGIN
    DECLARE i INT DEFAULT 0;
    DECLARE planetes VARCHAR(255) DEFAULT
'Tatooine,Coruscant,Naboo,Hoth,Endor,Dagobah,Mustafar,Alderaan,Bespin,Jakku';
    DECLARE priorites VARCHAR(255) DEFAULT 'basse,normale,haute,urgente';

    -- Désactiver les checks pour accélérer l'insertion
    SET autocommit = 0;
    SET unique_checks = 0;
    SET foreign_key_checks = 0;

    WHILE i < nb_lignes DO
        INSERT INTO transmissions (code_transmission, date_emission, emetteur, receuteur,
planete_origine, priorite, contenu)
            VALUES (
                CONCAT('TR-', LPAD(i, 10, '0')),
                DATE_SUB(NOW(), INTERVAL FLOOR(RAND() * 3650) DAY) + INTERVAL FLOOR(RAND() *
86400) SECOND,
                CONCAT('Agent-', FLOOR(RAND() * 10000)),
                CONCAT('Base-', FLOOR(RAND() * 1000)),
                ELT(FLOOR(1 + RAND() * 10), 'Tatooine', 'Coruscant', 'Naboo', 'Hoth', 'Endor',
'Dagobah', 'Mustafar', 'Alderaan', 'Bespin', 'Jakku'),
                ELT(FLOOR(1 + RAND() * 4), 'basse', 'normale', 'haute', 'urgente'),
                CONCAT('Message intercepté numéro ', i)
            );

        SET i = i + 1;

        -- Commit toutes les 10000 lignes
        IF i % 10000 = 0 THEN
            COMMIT;
        END IF;
    END WHILE;
END //
```

```

        END IF;
    END WHILE;

    COMMIT;

    -- Réactiver les checks
    SET unique_checks = 1;
    SET foreign_key_checks = 1;
    SET autocommit = 1;
END //

DELIMITER ;

-- Exécution (peut prendre plusieurs minutes)
CALL generer_transmissions(5000000);

```

Test sans index

```

-- Vérifier qu'il n'y a pas d'index (à part la PRIMARY KEY)
SHOW INDEX FROM transmissions;

-- Analyser la requête
EXPLAIN SELECT id, date_emission, emetteur
FROM transmissions
WHERE planete_origine = 'Tatooine' AND priorite = 'urgente';

-- Exécuter et mesurer le temps
SELECT id, date_emission, emetteur
FROM transmissions
WHERE planete_origine = 'Tatooine' AND priorite = 'urgente';

```

Résultat EXPLAIN (sans index) :

type	key	rows
ALL	NULL	5000000

Le SGBD parcourt les **5 millions de lignes** pour trouver les résultats. Temps estimé : plusieurs secondes.

Création d'un index composé

```
-- Créer un index sur les deux colonnes utilisées dans le WHERE
CREATE INDEX idx_planete_priorite ON transmissions(planete_origine, priorite);

-- Vérifier que l'index existe
SHOW INDEX FROM transmissions;
```

Test avec index

```
-- Analyser la même requête
EXPLAIN SELECT id, date_emission, emetteur
FROM transmissions
WHERE planete_origine = 'Tatooine' AND priorite = 'urgente';

-- Exécuter et comparer le temps
SELECT id, date_emission, emetteur
FROM transmissions
WHERE planete_origine = 'Tatooine' AND priorite = 'urgente';
```

Résultat EXPLAIN (avec index) :

	type	key	rows
ref		idx_planete_priorite	~125000

Le SGBD utilise l'index et ne parcourt qu'une fraction des lignes. Temps estimé : quelques millisecondes.

Configuration serveur

La configuration de MariaDB influence directement les performances.

Les paramètres se trouvent dans `/etc/mysql/mariadb.conf.d/50-server.cnf` (voir documentation pour une installation dockerisée).

innodb_buffer_pool_size

C'est le paramètre **le plus important** pour les performances. Il définit la quantité de mémoire allouée au cache des données et des index InnoDB.

```
# Règle : 70-80% de la RAM disponible sur un serveur dédié
# Exemple pour un serveur avec 8 Go de RAM
innodb_buffer_pool_size = 6G
```

Plus ce buffer est grand, plus les données fréquemment accédées restent en mémoire, évitant les lectures disque coûteuses.

max_connections

Nombre maximum de connexions simultanées autorisées.

```
# Valeur par défaut : 151
# Augmenter si vous avez beaucoup de clients simultanés
max_connections = 200
```

Chaque connexion consomme de la mémoire. Augmenter cette valeur sans augmenter la RAM peut causer des problèmes.

Optimisation des requêtes

Avant de chercher des solutions complexes, il faut s'assurer que les requêtes elles-mêmes sont bien écrites.

Éviter SELECT *

Récupérer toutes les colonnes consomme plus de ressources (mémoire, réseau) que nécessaire.

```
-- À éviter
SELECT * FROM transmissions WHERE planete_origine = 'Tatooine';

-- Préférer
SELECT id, date_emission, emetteur FROM transmissions WHERE planete_origine = 'Tatooine';
```

Limiter les résultats

Si vous n'avez besoin que des premiers résultats, utilisez `LIMIT`.

```
-- Récupérer les 100 dernières transmissions
SELECT id, date_emission, emetteur
FROM transmissions
ORDER BY date_emission DESC
LIMIT 100;
```

Éviter les fonctions sur les colonnes dans WHERE

Appliquer une fonction sur une colonne empêche l'utilisation des index.

```
-- À éviter : l'index sur date_emission ne sera PAS utilisé
SELECT * FROM transmissions WHERE YEAR(date_emission) = 2024;

-- Préférer : l'index peut être utilisé
SELECT * FROM transmissions WHERE date_emission BETWEEN '2024-01-01' AND '2024-12-31';
```

Préférer EXISTS à IN pour les sous-requêtes

Sur de gros volumes, `EXISTS` est généralement plus performant que `IN`.

```
-- Moins performant
SELECT * FROM personnages WHERE id IN (SELECT emetteur_id FROM transmissions WHERE priorite =
'urgente');

-- Plus performant
SELECT * FROM personnages p WHERE EXISTS (SELECT 1 FROM transmissions t WHERE t.emetteur_id =
p.id AND t.priorite = 'urgente');
```

Bonnes pratiques - Résumé

- **Requêtes** : éviter SELECT *, utiliser LIMIT, ne pas appliquer de fonctions sur les colonnes filtrées
- **Index** : indexer les colonnes utilisées dans WHERE, JOIN et ORDER BY
- **Ne pas sur-indexer** : chaque index ralentit les écritures
- **Analyser** : utiliser EXPLAIN régulièrement pour vérifier l'utilisation des index
- **Configurer** : adapter innodb_buffer_pool_size à la mémoire disponible

Revision #3

Created 15 January 2026 16:41:14 by Thibaud FRICHET

Updated 15 January 2026 16:47:46 by Thibaud FRICHET