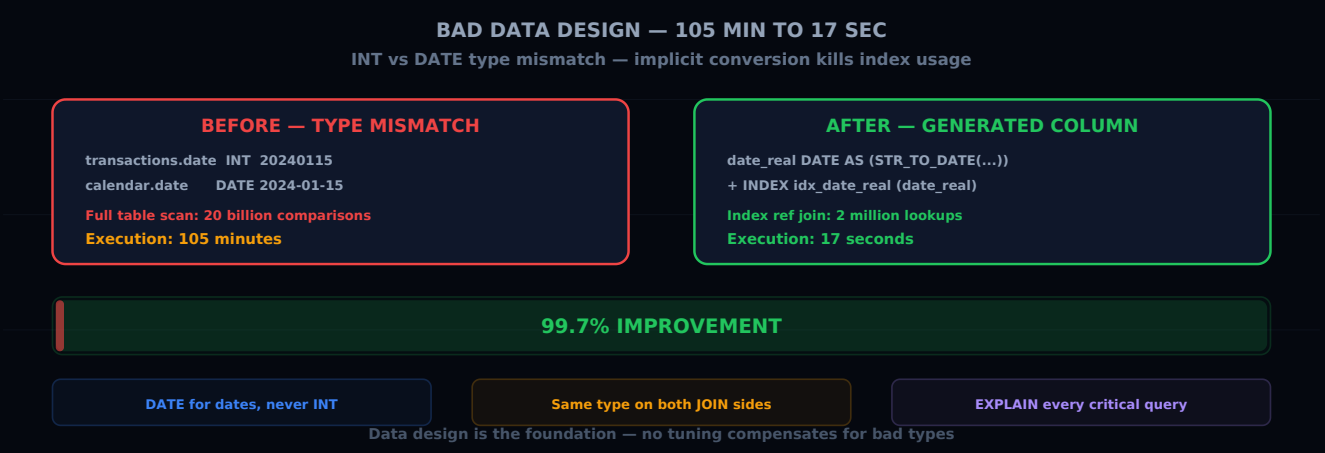
Un mauvais design de données mène à de mauvaises performances : de 105 minutes à 17 secondes

Sylvain ARBAUDIE · 23 juillet 2025

MARIADB PERFORMANCE OPTIMIZATION DATA-DESIGN



Le symptôme : 105 minutes pour une requête

Un client m'appelle en urgence. Leur batch nocturne, qui alimente les rapports journaliers, prend de plus en plus de temps. Ce qui tournait en 10 minutes il y a un an prend désormais **105 minutes**. La volumétrie a certes augmenté, mais pas au point de justifier une multiplication par dix du temps d'exécution.

La requête incriminée est un `JOIN` classique entre une table de transactions et une table de calendrier :

```
SELECT
    t.transaction_id,
    t.amount,
    t.transaction_date,
    c.fiscal_year,
    c.fiscal_quarter
FROM transactions t
JOIN calendar c ON t.transaction_date = c.calendar_date
WHERE t.created_at >= '2024-01-01';
```

Rien de remarquable en apparence. Deux tables, une jointure sur une date, un filtre temporel. Et pourtant, 105 minutes.

Le diagnostic : un mismatch de types

L'analyse du plan d'exécution (`EXPLAIN`) révèle un full table scan sur la table `calendar` . Étrange pour une jointure sur ce qui devrait être une clé primaire.

En examinant les structures des tables, le problème saute aux yeux :

```
-- Table transactions
CREATE TABLE transactions (
    transaction_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    amount DECIMAL(10,2),
    transaction_date INT NOT NULL,  -- ← stocké comme YYYYMMDD
    created_at DATETIME
);

-- Table calendar
CREATE TABLE calendar (
    calendar_date DATE NOT NULL PRIMARY KEY,
    fiscal_year SMALLINT,
    fiscal_quarter TINYINT
);
```

La colonne `transaction_date` dans la table `transactions` est un `INT` qui stocke la date au format `YYYYMMDD` (par exemple, `20240115` pour le 15 janvier 2024). La colonne `calendar_date` dans la table `calendar` est un vrai `DATE` .

Quand MariaDB / MySQL exécute le `JOIN` , il doit comparer un `INT` avec un `DATE` . Pour chaque ligne de `transactions` , le moteur convertit implicitement le `DATE` en `INT` (ou l'inverse) pour chaque ligne de `calendar` . Cette conversion implicite rend l'index sur `calendar_date` inutilisable. Résultat : un full table scan sur `calendar` pour chaque ligne de `transactions` .

Avec 2 millions de transactions et 10 000 lignes dans `calendar` , cela fait **20 milliards de comparaisons** avec conversion de type.

Pourquoi ne pas simplement changer le type ?

La réponse évidente serait de convertir la colonne `transaction_date` de `INT` en `DATE`. Mais dans la réalité des systèmes en production :

- La table fait 15 Go. Un `ALTER TABLE` prendrait des heures et verrouillerait la table.
- 47 procédures stockées et 12 vues référencent `transaction_date` comme `INT`.
- L'application PHP utilise des comparaisons arithmétiques sur cette colonne (`WHERE transaction_date > 20240101`).
- Le batch de chargement ETL envoie les dates au format `INT` depuis un système legacy.

Changer le type est la bonne solution à long terme, mais pas la solution immédiate dont le client a besoin ce soir.

La solution : une colonne virtuelle générée

MariaDB / MySQL supporte les colonnes virtuelles (ou generated columns). Ce sont des colonnes calculées dynamiquement à partir d'autres colonnes, sans stockage physique (VIRTUAL) ou avec stockage (STORED).

```
ALTER TABLE transactions
ADD COLUMN transaction_date_real DATE AS (
    STR_TO_DATE(CAST(transaction_date AS CHAR(8)), '%Y%m%d')
) VIRTUAL;
```

Cette colonne convertit l'`INT` en `DATE` à la volée. Mais une colonne virtuelle seule ne résout pas le problème de performance. Il faut un index :

```
ALTER TABLE transactions
ADD COLUMN transaction_date_real DATE AS (
    STR_TO_DATE(CAST(transaction_date AS CHAR(8)), '%Y%m%d')
) STORED,
ADD INDEX idx_transaction_date_real (transaction_date_real);
```

On utilise `STORED` plutôt que `VIRTUAL` pour pouvoir créer un index. La colonne est physiquement stockée et l'index est maintenu automatiquement lors des insertions et mises à jour.

La requête corrigée

```
SELECT
    t.transaction_id,
    t.amount,
    t.transaction_date,
    c.fiscal_year,
    c.fiscal_quarter
FROM transactions t
JOIN calendar c ON t.transaction_date_real = c.calendar_date
WHERE t.created_at >= '2024-01-01';
```

Le `JOIN` compare maintenant un `DATE` avec un `DATE`. L'index est utilisable. Le plan d'exécution montre un `ref` au lieu d'un full scan.

Le résultat : 17 secondes

Métrique	Avant	Après	Amélioration
Temps d'exécution	105 min	17 sec	99,7%
Lignes examinées	~20 milliards	~2 millions	99,99%
Type de scan	Full scan	Index ref	—

De 105 minutes à 17 secondes. Une amélioration de **99,7%** sans changer le schéma existant, sans modifier l'application, sans toucher aux procédures stockées.

Pourquoi les conversions implicites sont un piège

Ce cas illustre un problème fondamental : les conversions implicites de type dans les jointures et les clauses WHERE sont des tueurs de performance silencieux.

MariaDB / MySQL effectue des conversions implicites dans de nombreux cas :

- `INT` comparé à `VARCHAR` : l'INT est converti en VARCHAR
- `INT` comparé à `DATE` : la DATE est convertie en nombre
- `VARCHAR(utf8)` comparé à `VARCHAR(latin1)` : conversion de charset
- `DECIMAL` comparé à `FLOAT` : conversion en virgule flottante

Dans chaque cas, la conversion rend l'index inutilisable car le moteur ne peut pas faire de recherche directe dans un index B-tree si la valeur doit d'abord être transformée.

La leçon : le design des données est la fondation

Les performances d'une base de données se jouent au moment du design, pas au moment du tuning. Aucun index, aucune configuration de buffer pool, aucun hardware ne compensera un mauvais choix de type de données.

Les règles fondamentales :

1. **Une date doit être stockée comme `DATE` ou `DATETIME`**, jamais comme `INT` ou `VARCHAR`.
2. **Les colonnes de jointure doivent avoir le même type et le même charset/collation.**
3. **Utilisez `EXPLAIN` systématiquement** pour vérifier que vos jointures utilisent les index.
4. **Surveillez les conversions implicites** avec l'outil `EXPLAIN ANALYZE` (MariaDB 10.1+).

Le data design n'est pas glamour. Ce n'est pas aussi excitant que le tuning de variables système ou la mise en place d'un cluster Galera. Mais c'est la fondation. Et quand la fondation est mauvaise, tout le reste s'écroule — 105 minutes à la fois.

Cet article a été initialement publié sur Medium.