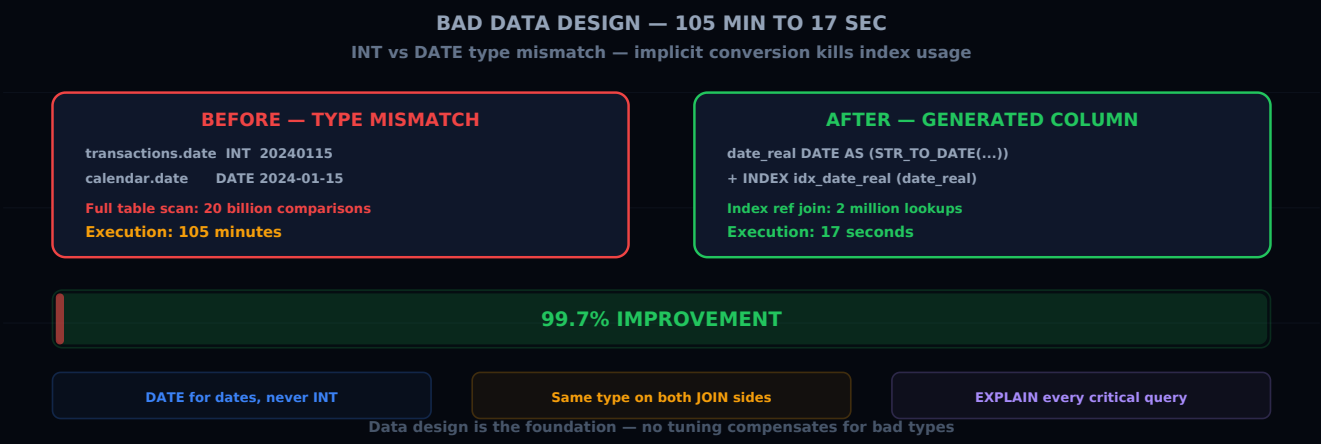
Bad Data Design Leads to Poor Performances: From 105 Minutes to 17 Seconds

Sylvain ARBAUDIE · July 23, 2025

MARIADB PERFORMANCE OPTIMIZATION DATA-DESIGN



The Symptom: 105 Minutes for a Query

A client calls me urgently. Their nightly batch, which feeds daily reports, is taking longer and longer. What used to run in 10 minutes a year ago now takes **105 minutes**. Volume has increased, certainly, but not enough to justify a tenfold increase in execution time.

The offending query is a classic `JOIN` between a transactions table and a calendar table:

```
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';
```

Nothing remarkable at first glance. Two tables, a join on a date, a time filter. And yet, 105 minutes.

The Diagnosis: A Type Mismatch

The execution plan analysis (`EXPLAIN`) reveals a full table scan on the `calendar` table. Strange for a join on what should be a primary key.

Examining the table structures, the problem jumps out:

```
-- transactions table
CREATE TABLE transactions (
    transaction_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    amount DECIMAL(10,2),
    transaction_date INT NOT NULL,  -- stored as YYYYMMDD
    created_at DATETIME
);

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

The `transaction_date` column in the `transactions` table is an `INT` storing the date in `YYYYMMDD` format (for example, `20240115` for January 15, 2024). The `calendar_date` column in the `calendar` table is an actual `DATE` .

When MariaDB / MySQL executes the `JOIN` , it must compare an `INT` with a `DATE` . For each row in `transactions` , the engine implicitly converts the `DATE` to an `INT` (or vice versa) for every row in `calendar` . This implicit conversion makes the index on `calendar_date` unusable. Result: a full table scan on `calendar` for every row in `transactions` .

With 2 million transactions and 10,000 rows in `calendar` , that is **20 billion comparisons** with type conversion.

Why Not Simply Change the Type?

The obvious answer would be to convert the `transaction_date` column from `INT` to `DATE` . But in the reality of production systems:

- The table is 15 GB. An `ALTER TABLE` would take hours and lock the table.

- 47 stored procedures and 12 views reference `transaction_date` as `INT`.
- The PHP application uses arithmetic comparisons on this column (`WHERE transaction_date > 20240101`).
- The ETL loading batch sends dates in `INT` format from a legacy system.

Changing the type is the right long-term solution, but not the immediate fix the client needs tonight.

The Solution: A Virtual Generated Column

MariaDB / MySQL supports virtual columns (or generated columns). These are columns dynamically computed from other columns, without physical storage (VIRTUAL) or with storage (STORED).

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

This column converts the `INT` to `DATE` on the fly. But a virtual column alone does not solve the performance problem. An index is needed:

```
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);
```

We use `STORED` rather than `VIRTUAL` to be able to create an index. The column is physically stored and the index is automatically maintained during inserts and updates.

The Corrected Query

```
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';
```

The `JOIN` now compares a `DATE` with a `DATE`. The index is usable. The execution plan shows a `ref` instead of a full scan.

The Result: 17 Seconds

Metric	Before	After	Improvement
Execution time	105 min	17 sec	99.7%
Rows examined	~20 billion	~2 million	99.99%
Scan type	Full scan	Index ref	—

From 105 minutes to 17 seconds. A **99.7% improvement** without changing the existing schema, without modifying the application, without touching stored procedures.

Why Implicit Conversions Are a Trap

This case illustrates a fundamental problem: implicit type conversions in joins and WHERE clauses are silent performance killers.

MariaDB / MySQL performs implicit conversions in many cases:

- `INT` compared to `VARCHAR`: the INT is converted to VARCHAR
- `INT` compared to `DATE`: the DATE is converted to a number
- `VARCHAR(utf8)` compared to `VARCHAR(latin1)`: charset conversion
- `DECIMAL` compared to `FLOAT`: floating-point conversion

In each case, the conversion makes the index unusable because the engine cannot do a direct B-tree index lookup if the value must first be transformed.

The Lesson: Data Design Is the Foundation

Database performance is determined at design time, not at tuning time. No index, no buffer pool configuration, no hardware will compensate for a bad data type choice.

The fundamental rules:

1. **A date must be stored as `DATE` or `DATETIME`**, never as `INT` or `VARCHAR`.
2. **Join columns must have the same type and the same charset/collation.**
3. **Use `EXPLAIN` systematically** to verify that your joins use indexes.
4. **Watch for implicit conversions** with the `EXPLAIN ANALYZE` tool (MariaDB 10.1+).

Data design is not glamorous. It is not as exciting as tuning system variables or setting up a Galera cluster. But it is the foundation. And when the foundation is bad, everything else crumbles — 105 minutes at a time.

This article was originally published on Medium.