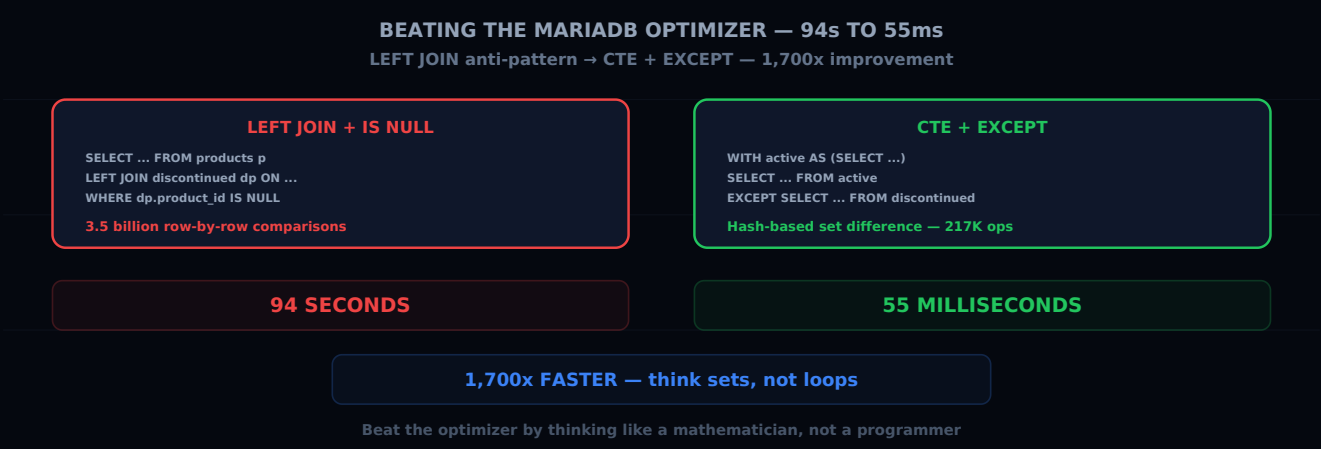
I Beat the MariaDB Optimizer: From 94 Seconds to 55 Milliseconds

Sylvain ARBAUDIE · July 15, 2025

MARIADB OPTIMIZER PERFORMANCE SQL



The Context: MySQL 8 to MariaDB 11.4 Migration

A migration project from MySQL 8 to MariaDB 11.4 is going well overall. Functional tests are green, performance tests too — except for one query. A single query that ran in 2 seconds on MySQL 8 and now takes **94 seconds** on MariaDB 11.4.

This is a classic migration regression: the MySQL and MariaDB optimizers have diverged significantly since the fork. An execution plan that worked well on one can be catastrophic on the other.

The Problematic Query

The original query uses the classic LEFT JOIN pattern to find records that do NOT exist in another table:

```
SELECT p.product_id, p.product_name, p.category_id
FROM products p
LEFT JOIN discontinued_products dp
    ON p.product_id = dp.product_id
```

```
    AND p.category_id = dp.category_id
WHERE dp.product_id IS NULL
  AND p.status = 'active'
  AND p.created_at >= '2023-01-01';
```

The intent is clear: find all active products that do not appear in the discontinued products table. It is an "anti-join" pattern implemented via LEFT JOIN + IS NULL.

Why 94 Seconds?

Analyzing the execution plan on MariaDB 11.4 reveals the problem. The optimizer chooses to:

1. Scan the `products` table using the index on `status` (200,000 rows)
2. For each row, scan the `discontinued_products` table to verify the absence of a match

With a `discontinued_products` table of 17,500 rows, this represents approximately **3.5 billion comparisons**. The MySQL 8 optimizer chose a different plan with a hash join, much more efficient for this pattern.

The fundamental problem is not the MariaDB optimizer — it is the query itself. LEFT JOIN + IS NULL to implement an anti-join is a historical anti-pattern dating from the era when SQL had no better alternatives.

The Solution: CTE + EXCEPT

MariaDB has supported Common Table Expressions (CTE) since version 10.2 and the `EXCEPT` operator since version 10.3. These two features allow rewriting the logic in a much more explicit way:

```
WITH active_products AS (
    SELECT product_id, category_id
    FROM products
    WHERE status = 'active'
      AND created_at >= '2023-01-01'
),
still_active AS (
    SELECT product_id, category_id FROM active_products
    EXCEPT
    SELECT product_id, category_id FROM discontinued_products
```

```
)
SELECT p.product_id, p.product_name, p.category_id
FROM products p
JOIN still_active sa
    ON p.product_id = sa.product_id
    AND p.category_id = sa.category_id;
```

Why It Is Faster

1. **The `active_products` CTE** materializes the 200,000 active products in memory. A single scan of the `products` table.
2. **The `EXCEPT` operator** performs a set operation: it takes the set of active products and subtracts those appearing in `discontinued_products`. It is a hash-based set difference, not a row-by-row comparison.
3. **The final JOIN** with the `still_active` CTE is a simple lookup on an already filtered set.

The Result: 55 Milliseconds

Metric	LEFT JOIN	CTE + EXCEPT	Improvement
Time	94 seconds	55 ms	1,700x
Comparisons	~3.5 billion	~217,500	99.99%
Approach	Row by row	Set-based	—

From 94 seconds to 55 milliseconds. A factor of **1,700**. Not by changing a configuration parameter. Not by adding an index. By **rethinking the query logic**.

The LEFT JOIN Anti-Pattern: Why It Persists

The LEFT JOIN + IS NULL pattern for anti-joins is everywhere. You find it in tutorials, online courses, Stack Overflow answers. It persists for several reasons:

Historical: Before SQL:1999, there was no `EXCEPT`, no optimized `NOT EXISTS`, no CTE. LEFT JOIN + IS NULL was the only portable option.

Habit: Developers learn a pattern that works and reuse it. "It works" is the enemy of "it is optimal."

Compatibility: LEFT JOIN works on all versions of all databases. `EXCEPT` is only supported by recent versions.

Alternatives to Know

For anti-joins, three alternatives exist:

NOT EXISTS (often the best choice)

```
SELECT p.product_id, p.product_name, p.category_id
FROM products p
WHERE p.status = 'active'
  AND p.created_at >= '2023-01-01'
  AND NOT EXISTS (
    SELECT 1 FROM discontinued_products dp
    WHERE dp.product_id = p.product_id
      AND dp.category_id = p.category_id
  );
```

`NOT EXISTS` is generally well optimized by both MariaDB and MySQL engines. The optimizer can use an inverted semi-join, much more efficient than a LEFT JOIN.

NOT IN (beware of NULLs)

```
SELECT product_id, product_name, category_id
FROM products
WHERE status = 'active'
  AND created_at >= '2023-01-01'
  AND (product_id, category_id) NOT IN (
    SELECT product_id, category_id
    FROM discontinued_products
  );
```

Caution: `NOT IN` has tricky behavior with NULL values. If `discontinued_products.product_id` can be NULL, NOT IN semantics return an empty result. Always use NOT EXISTS if NULLs are possible.

EXCEPT (the most readable)

```
SELECT product_id, category_id FROM products
WHERE status = 'active' AND created_at >= '2023-01-01'
EXCEPT
SELECT product_id, category_id FROM discontinued_products;
```

`EXCEPT` is the purest expression of the set operation "A minus B." But it only returns the columns of the operation, not additional columns — hence the use of a CTE to reintroduce missing columns via a JOIN.

The Lesson

The optimizer is a tool, not a miracle worker. When a query is slow, the first question should not be "what hint can I add?" or "what parameter should I change?" The first question should be: **does my query correctly express my intent?**

A LEFT JOIN + IS NULL expresses "join then filter non-matches." An `EXCEPT` directly expresses "subtract this set from that other set." The second formulation is closer to the business intent, and it gives the optimizer a much better chance of finding an efficient plan.

Beat the optimizer by thinking like a mathematician, not a programmer.

This article was originally published on Medium.