Got it — you want **only rows from TableA** where the key\_col **does not exist in TableB**, and performance is an issue because the tables are large.

Let’s focus on:

* ✅ Correctness
* 🚀 Performance (with real-world SQL Server optimizations)

**🔍 1. Correct and Optimized Query**

-- Get rows from TableA where key\_col is NOT in TableB

SELECT A.\*

FROM TableA A

WHERE NOT EXISTS (

SELECT 1

FROM TableB B

WHERE B.key\_col = A.key\_col

);

NOT EXISTS is preferred over NOT IN for large tables and avoids NULL-related surprises.

**🚀 2. Performance Optimization Techniques**

**✅ a) Add Proper Indexes**

Make sure key\_col is indexed in both tables:

CREATE NONCLUSTERED INDEX idx\_TableA\_key\_col ON TableA(key\_col);

CREATE NONCLUSTERED INDEX idx\_TableB\_key\_col ON TableB(key\_col);

* This helps SQL Server *quickly* locate matching keys.
* Especially important for NOT EXISTS to use an **anti-semi join** efficiently.

**✅ b) Use LEFT JOIN ... IS NULL (may be faster depending on plan)**

This can sometimes outperform NOT EXISTS:

SELECT A.\*

FROM TableA A

LEFT JOIN TableB B ON A.key\_col = B.key\_col

WHERE B.key\_col IS NULL;

* Let SQL Server’s optimizer choose the best join method.
* Sometimes the execution plan with this approach leads to **hash joins** which can be faster for large datasets.

**✅ c) Use OPTION (HASH JOIN) or LOOP JOIN (if needed)**

Force an efficient join if SQL Server chooses a bad plan:

SELECT A.\*

FROM TableA A

LEFT JOIN TableB B ON A.key\_col = B.key\_col

WHERE B.key\_col IS NULL

OPTION (HASH JOIN);

* Test with HASH JOIN and MERGE JOIN — pick whichever runs fastest.

**✅ d) Eliminate Blocking Operators**

If you're doing anything like DISTINCT, ORDER BY, GROUP BY, or UNION — **remove them** if they’re not absolutely required. They cause **blocking operations** that slow down queries.

**✅ e) Consider a Temporary or Indexed Staging Table for TableB**

If TableB is queried a lot, or is very slow to scan:

SELECT key\_col INTO #TableB\_Keys

FROM TableB

-- Add index

CREATE NONCLUSTERED INDEX idx\_temp\_key ON #TableB\_Keys(key\_col);

Then:

SELECT A.\*

FROM TableA A

WHERE NOT EXISTS (

SELECT 1 FROM #TableB\_Keys B WHERE B.key\_col = A.key\_col

);

This can dramatically improve performance for repeated or complex queries.

**✅ f) Update Statistics and Rebuild Indexes**

If you're seeing long query times, try:

-- Update stats

UPDATE STATISTICS TableA;

UPDATE STATISTICS TableB;

-- Optional: Rebuild fragmented indexes (especially after many updates/inserts)

ALTER INDEX ALL ON TableA REBUILD;

ALTER INDEX ALL ON TableB REBUILD;

If you’d like, I can help analyze your **actual execution plan** — just copy the XML or graphical plan here (or upload a screenshot). Want to go there next?