DISTINCT TRXCOUNT

It looks like the issue stems from how the TRXCOUNT subquery is joined to the main query. Since the TRXCOUNT is grouped by LOCATION.LocaName and SALESREC.DateFrom, it might be expanding beyond the intended scope due to the multiple rows in SALESREC matching the same ReferDoc. This can cause TrxCount to be calculated incorrectly because of the additional joins that might be multiplying rows, especially if SALESDTL or other tables are causing a mismatch in the rows.

**Solution Approach:**

1. **Ensure Proper Grouping in the TRXCOUNT CTE**: We need to ensure that TRXCOUNT calculates the transaction count correctly for each ReferDoc, not expanding based on the joined tables. It should focus solely on ReferDoc and the Location.
2. **Control the Join with TRXCOUNT**: The join to TRXCOUNT should maintain one row per SALESREC.Location and SALESREC.DateFrom without causing duplicates when joining to SALESDTL.
3. **Calculate TrxCount independently from SALESDTL**: Since you're using ReferDoc for transaction count, we can isolate it and avoid it being impacted by the SALESDTL table.

Here is an updated version of your query:

WITH TRXCOUNT AS (

SELECT

LOCATION.LocaName,

SALESREC.DateFrom AS Date\_\_\_\_,

COUNT(DISTINCT SALESREC.ReferDoc) AS TrxCount -- Count DISTINCT ReferDoc to avoid duplications

FROM SALESREC

JOIN LOCATION ON SALESREC.Location = LOCATION.Location

WHERE SALESREC.Disabled = 0

AND SALESREC.DateFrom >= @dDateFrom

AND SALESREC.DateFrom <= @dDateTo\_\_

AND SALESREC.Location LIKE @cLocation

GROUP BY LOCATION.LocaName, SALESREC.DateFrom

)

SELECT

LOCATION.StoreGrp,

LOCATION.LocaName,

SALESREC.DateFrom AS Date\_\_\_\_,

TRXCOUNT.TrxCount,

SUM(SALESDTL.Quantity) AS TotalQty,

SUM(SALESDTL.Quantity \* SALESDTL.ItemPrce) AS TotalPrc,

SUM(SALESDTL.Quantity \* SALESDTL.ItemPrce) - SUM(SALESDTL.Quantity \* SALESDTL.Amount\_\_) AS TotalDsc,

SUM(SALESDTL.Quantity \* SALESDTL.Amount\_\_) AS TotalAmt,

SUM(SALESDTL.Quantity \* SALESDTL.LandCost) AS TotalCos

FROM SALESDTL

JOIN SALESREC ON SALESDTL.CtrlNum\_ = SALESREC.CtrlNum\_

JOIN LOCATION ON SALESREC.Location = LOCATION.Location

JOIN TRXCOUNT ON LOCATION.LocaName = TRXCOUNT.LocaName AND SALESREC.DateFrom = TRXCOUNT.Date\_\_\_\_

WHERE SALESREC.Disabled = 0

AND SALESREC.DateFrom >= @dDateFrom

AND SALESREC.DateFrom <= @dDateTo\_\_

AND SALESREC.Location LIKE @cLocation

GROUP BY LOCATION.StoreGrp, LOCATION.LocaName, SALESREC.DateFrom, TRXCOUNT.TrxCount

**Key Changes:**

1. **Count DISTINCT ReferDoc**: In the TRXCOUNT CTE, I changed the COUNT(SALESREC.ReferDoc) to COUNT(DISTINCT SALESREC.ReferDoc) to ensure that only unique transaction counts per ReferDoc are considered. This avoids overcounting if there are multiple rows per ReferDoc.
2. **Join Condition on Date**: When joining TRXCOUNT to the main query, I added SALESREC.DateFrom = TRXCOUNT.Date\_\_\_\_ to make sure the count is correctly associated with the correct date.
3. **Maintain Grouping and Aggregations**: The rest of the query structure remains the same, with proper grouping for TotalQty, TotalPrc, etc., and no changes to how SALESDTL is processed.

**Why This Should Work:**

* **Transaction count (TrxCount)**: By ensuring COUNT(DISTINCT SALESREC.ReferDoc) in TRXCOUNT, we avoid inflating the transaction count due to multiple matching rows in SALESDTL or other tables.
* **Joins**: The join to TRXCOUNT is now more controlled, based on both LocaName and DateFrom, to avoid expansion issues.
* **No Duplication**: The changes ensure that you are not inadvertently multiplying rows when joining with SALESDTL, thus preserving the accuracy of TrxCount.

Let me know if that works or if you need further adjustments!

REQUIRED INDEX

I'm glad it worked! Now, to optimize the query performance, indexing plays a crucial role, especially when working with large datasets. Given the structure of your query and the tables involved, here are the key fields that should be indexed to help with query speed:

**1. SALESREC Table:**

This table is joined on the following columns:

* **Location**: This is part of a join condition (SALESREC.Location = LOCATION.Location), so it should be indexed to speed up the lookup of records for a specific location.
  + **Recommended Index**: INDEX(Location)
* **DateFrom**: This is used for filtering (SALESREC.DateFrom >= @dDateFrom AND SALESREC.DateFrom <= @dDateTo\_\_) and also in the TRXCOUNT CTE for grouping.
  + **Recommended Index**: INDEX(DateFrom)
* **ReferDoc**: In the TRXCOUNT CTE, COUNT(DISTINCT SALESREC.ReferDoc) is used. Indexing ReferDoc helps speed up the count and distinct filtering.
  + **Recommended Index**: INDEX(ReferDoc)
* **CtrlNum\_**: This column is used to join SALESDTL and SALESREC (SALESDTL.CtrlNum\_ = SALESREC.CtrlNum\_).
  + **Recommended Index**: INDEX(CtrlNum\_)

**Composite Index Suggestion for SALESREC:**

If your query commonly filters by both Location and DateFrom, a composite index on (Location, DateFrom) might be helpful to speed up the filtering process.

* **Composite Index**: INDEX(Location, DateFrom)

This composite index allows for efficient filtering on both Location and DateFrom simultaneously.

**2. LOCATION Table:**

The LOCATION table is used in several parts of the query:

* **Location**: It's used in the join condition SALESREC.Location = LOCATION.Location.
  + **Recommended Index**: INDEX(Location)
* **LocaName**: It's used in the TRXCOUNT CTE for grouping, and also in the join with TRXCOUNT.
  + **Recommended Index**: INDEX(LocaName)
* **StoreGrp**: This column is part of the final GROUP BY clause.
  + **Recommended Index**: INDEX(StoreGrp)

**Composite Index Suggestion for LOCATION:**

If the queries often involve filtering by Location and LocaName, a composite index on (Location, LocaName) could be useful. This will help the query engine quickly find the correct rows.

* **Composite Index**: INDEX(Location, LocaName)

**3. SALESDTL Table:**

This table is joined using CtrlNum\_ with SALESREC and used in aggregation.

* **CtrlNum\_**: This is part of the join condition (SALESDTL.CtrlNum\_ = SALESREC.CtrlNum\_).
  + **Recommended Index**: INDEX(CtrlNum\_)
* **Quantity**: Used in the SUM() aggregations.
  + **Recommended Index**: If Quantity is often filtered or grouped, you might want to index it. But if it's only being summed in this query and isn't filtered or grouped elsewhere, this is less critical for indexing.
* **ItemPrce** and **Amount\_\_**: Both used in calculations (Quantity \* ItemPrce, Quantity \* Amount\_\_).
  + **Recommended Index**: These fields are used in arithmetic operations, and while indexing them may not improve performance significantly (since you're doing aggregate functions), if SALESDTL contains large amounts of data, indexing ItemPrce and Amount\_\_ may help with faster access in calculations.

**4. TRXCOUNT CTE:**

Although TRXCOUNT is a common table expression (CTE) and isn't a physical table, it essentially acts as a derived table. The performance of CTEs typically depends on how well the underlying tables are indexed.

* **LocaName**: Used in the GROUP BY clause and for joining TRXCOUNT with LOCATION.
  + **Recommended Index**: If not already indexed, INDEX(LocaName) on the LOCATION table should help.

**5. General Indexing Best Practices:**

* **Date Ranges**: If you frequently filter based on date ranges (like DateFrom), it’s helpful to have a non-clustered index on DateFrom. This speeds up the filter condition in the query.
* **Avoid Indexing All Columns**: Don’t over-index. Indexing too many columns can slow down write operations (INSERT, UPDATE, DELETE), so try to index only the columns you frequently filter or join on.
* **Covering Indexes**: If you find that the query is still slow, consider creating a covering index that includes all the columns used in WHERE, JOIN, GROUP BY, and SELECT. A covering index can speed up the query by avoiding table lookups.

**Example Index Creation:**

Here’s an example of how you could create indexes for the key fields:

-- Index on Location in SALESREC for filtering and joining

CREATE INDEX idx\_salesrec\_location ON SALESREC(Location);

-- Index on DateFrom in SALESREC for filtering and grouping

CREATE INDEX idx\_salesrec\_datefrom ON SALESREC(DateFrom);

-- Index on ReferDoc in SALESREC for distinct counting

CREATE INDEX idx\_salesrec\_referdoc ON SALESREC(ReferDoc);

-- Composite index on Location and DateFrom in SALESREC

CREATE INDEX idx\_salesrec\_location\_datefrom ON SALESREC(Location, DateFrom);

-- Index on CtrlNum\_ in SALESDTL for joining with SALESREC

CREATE INDEX idx\_salesdtl\_ctrlnum ON SALESDTL(CtrlNum\_);

-- Index on Location in LOCATION for joining

CREATE INDEX idx\_location\_location ON LOCATION(Location);

-- Index on LocaName in LOCATION for grouping and joining with TRXCOUNT

CREATE INDEX idx\_location\_locaname ON LOCATION(LocaName);

-- Composite index on Location and LocaName in LOCATION (if frequently filtered together)

CREATE INDEX idx\_location\_location\_locaname ON LOCATION(Location, LocaName);

-- Index on StoreGrp in LOCATION for grouping

CREATE INDEX idx\_location\_storegrp ON LOCATION(StoreGrp);

**Conclusion:**

1. **Focus on the join columns** (Location, CtrlNum\_, LocaName, etc.) and the **filtering columns** (DateFrom, ReferDoc) for indexing.
2. **Composite indexes** can improve performance when multiple columns are frequently filtered or grouped together.
3. **Avoid over-indexing** to prevent unnecessary overhead on insert/update operations.

These indexing strategies should help in significantly improving the query performance. You might need to test and fine-tune the indexes depending on your specific data distribution and query patterns.

Let me know if you need more details on creating or testing these indexes!