Your SQL script seems quite comprehensive, but there are several improvements that can be made in terms of SQL performance, structure, and parameterization to make it more modular and easier to maintain. Let's break it down into a few key areas of improvement:

**1. Parameterization:**

* You're currently using SQL LIKE with @cBrandNum and @cLocation, but it looks like the parameters are intended to filter by exact matches (e.g., BrandNum and Location). Using LIKE in this case may not be the best approach if you're looking for exact matches.
* We can add placeholders for all parameters (@cBrandNum, @cLocation, @dDateAsOf), ensuring the query is parameterized correctly.

**2. SQL Query Formatting:**

* The SQL query is currently formatted as one long string, which is hard to maintain. Breaking it down into smaller sections (especially the SELECT statements for each part of your union) will make it cleaner and more maintainable.
* It seems that each UNION ALL query is repeating a lot of the same logic. Extracting common filters into a subquery or a JOIN could reduce redundancy.

**3. Avoiding Redundancy in UNION ALL:**

* You're repeatedly filtering on LOCATION.Damaged\_ = 0, ITEMLIST.Services = 0, and ITEMLIST.Outright = 1. You could create a subquery or a JOIN for these filters, and then perform the summation in the main query. This avoids repeating the same conditions across multiple UNION blocks.

**4. SQL Formatting for Readability:**

* Proper indentation and line breaks can help readability, especially for complex queries like this.

**5. Additional Improvements for Flexibility:**

* If you need the query to accept other parameters (e.g., specific date ranges or other criteria), you can make it flexible by dynamically adding filters based on the request parameters.

**Refactored Code with Parameters and Optimized Query**

Here’s how you can refactor the code for better readability, performance, and flexibility:

const { queryDatabase } = require('../../DBConnect/dbConnect');

const StockEndingByLocation = async (req, res) => {

const cBrandNum = req.query.BrandNum;

const cLocation = req.query.Location;

const dDateAsOf = req.query.DateAsOf;

// Base SQL query structure

let cSql = `

WITH CommonFilters AS (

SELECT

LOCATION.LocaName,

ITEMLIST.ItemCode,

ITEMLIST.BrandNum,

LOCATION.Location

FROM LOCATION

JOIN ITEMLIST ON ITEMLIST.BrandNum = @cBrandNum

WHERE

LOCATION.Location LIKE @cLocation

AND LOCATION.Damaged\_ = 0

AND LOCATION.Disabled = 0

AND ITEMLIST.Services = 0

AND ITEMLIST.Outright = 1

)

SELECT L.LocaName, SUM(CASE WHEN T.Type = 'Count' THEN T.Quantity ELSE 0 END) AS Quantity

FROM (

-- Count Movement

SELECT

CF.LocaName,

(COUNTDTL.Quantity + COUNTDTL.AdjusCnt) AS Quantity,

'Count' AS Type

FROM COUNTDTL

JOIN COUNTREC ON COUNTDTL.CtrlNum\_ = COUNTREC.CtrlNum\_

JOIN CommonFilters CF ON CF.ItemCode = COUNTDTL.ItemCode AND CF.Location = COUNTREC.Location

WHERE COUNTREC.Date\_\_\_\_ = CF.BeginDte

GROUP BY CF.LocaName

UNION ALL

-- Sales Movement

SELECT

CF.LocaName,

-SALESDTL.Quantity AS Quantity,

'Sales' AS Type

FROM SALESDTL

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

JOIN CommonFilters CF ON CF.ItemCode = SALESDTL.ItemCode AND CF.Location = SALESREC.Location

WHERE SALESDTL.Date\_\_\_\_ BETWEEN CF.BeginDte + 1 AND @dDateAsOf

GROUP BY CF.LocaName

UNION ALL

-- Purchases Movement

SELECT

CF.LocaName,

PURCHDTL.Quantity AS Quantity,

'Purchase' AS Type

FROM PURCHDTL

JOIN PURCHREC ON PURCHDTL.CtrlNum\_ = PURCHREC.CtrlNum\_

JOIN CommonFilters CF ON CF.ItemCode = PURCHDTL.ItemCode AND CF.Location = PURCHREC.Location

WHERE PURCHREC.Date\_\_\_\_ BETWEEN CF.BeginDte + 1 AND @dDateAsOf

GROUP BY CF.LocaName

UNION ALL

-- Adjustments Movement

SELECT

CF.LocaName,

ITEMADJU.Quantity AS Quantity,

'Adjustment' AS Type

FROM ITEMADJU

JOIN ADJUSREC ON ADJUSREC.CtrlNum\_ = ITEMADJU.CtrlNum\_

JOIN CommonFilters CF ON CF.ItemCode = ITEMADJU.ItemCode AND CF.Location = ADJUSREC.Location

WHERE ADJUSREC.Date\_\_\_\_ BETWEEN CF.BeginDte + 1 AND @dDateAsOf

GROUP BY CF.LocaName

UNION ALL

-- Stock Movement (Outwards)

SELECT

CF.LocaName,

-STOCKDTL.Quantity AS Quantity,

'StockOut' AS Type

FROM STOCKDTL

JOIN STOCKREC ON STOCKREC.CtrlNum\_ = STOCKDTL.CtrlNum\_

JOIN CommonFilters CF ON CF.ItemCode = STOCKDTL.ItemCode AND CF.Location = STOCKREC.WhseFrom

WHERE STOCKREC.Date\_\_\_\_ BETWEEN CF.BeginDte + 1 AND @dDateAsOf

GROUP BY CF.LocaName

UNION ALL

-- Stock Movement (Inwards)

SELECT

CF.LocaName,

STOCKDTL.QtyRecvd AS Quantity,

'StockIn' AS Type

FROM STOCKDTL

JOIN STOCKREC ON STOCKREC.CtrlNum\_ = STOCKDTL.CtrlNum\_

JOIN CommonFilters CF ON CF.ItemCode = STOCKDTL.ItemCode AND CF.Location = STOCKREC.WhseTo\_\_

WHERE STOCKREC.Date\_\_\_\_ BETWEEN CF.BeginDte + 1 AND @dDateAsOf

GROUP BY CF.LocaName

) AS T

GROUP BY L.LocaName

`;

// Parameters object

const params = {

cBrandNum,

cLocation,

dDateAsOf

};

try {

// Execute query

const result = await queryDatabase(cSql, params);

res.json(result);

} catch (err) {

console.error('Database query error:', err.message); // Log the error message

res.status(500).send('Error fetching inventory data');

}

};

module.exports = { StockEndingByLocation };

**Key Changes:**

1. **Common Filters Using WITH Clause:**
   * The common filters for BrandNum, Location, and other conditions (like Damaged\_, Disabled, Services, Outright) are extracted into a WITH clause. This makes the query DRY (Don't Repeat Yourself), as you don't need to repeat the same conditions in every UNION ALL block.
2. **Parameterization:**
   * We are now using placeholders (@cBrandNum, @cLocation, and @dDateAsOf) for the dynamic parameters that the user provides.
3. **Cleaner SQL Structure:**
   * The UNION ALL blocks are now cleanly structured with proper indentation, making the query easier to read and maintain.
4. **Group By and Aggregation:**
   * After all the individual movements are aggregated into the subquery (T), we use GROUP BY L.LocaName in the final selection to get the sum of quantities per location.

**Flexibility:**

* This refactored query is more flexible and will allow you to easily add additional parameters or conditions without rewriting the entire query.
* You can modify the CommonFilters or add more filters to the SELECT block for more custom reporting.

**Next Steps:**

* Ensure that the queryDatabase function is securely handling parameterized queries.
* Test with different parameters to ensure the expected output.

Let me know if you'd like to refine it further or have any questions!