In order to improve the performance of a slow SQL SELECT query involves several steps and considerations.

First, you should use the EXPLAIN statement to get the execution plan and identify bottlenecks in the execution plan. Then, address the identified issues**.**

Additionally**,** look for selections like **"SELECT ";* remove them and specify only the columns needed.

Ensure that columns used in WHERE, JOIN, ORDER BY, and GROUP BY clauses have appropriate indexes.

Look out for calculations on indexed columns, as the index might not be used.

Here is a comprehensive route to solving this problem:

1. **Analyze the Query Execution Plan**

The first step is to understand how the query is being executed by the database engine.

Execution Plan: Use the EXPLAIN statement (in MySQL, PostgreSQL) or SET STATISTICS PROFILE ON (in SQL Server) to get the execution plan of the query.

Identify Bottlenecks: Look for full table scans, large joins, and sorts that might be causing delays.

2. **Optimize Query Structure**

Select Only Necessary Columns: Avoid using SELECT *. Instead, specify only the columns you need.

Use Indexes: Ensure that columns used in WHERE, JOIN, ORDER BY, and GROUP BY clauses have appropriate indexes.

Avoid Calculations on Indexed Columns: If you perform calculations on indexed columns, the index might not be used.

3. **Indexing**

Create Indexes: Add indexes on columns that are frequently used in search conditions, joins, and sorting.

Composite Indexes: Use composite indexes for queries that filter on multiple columns.

Index Maintenance: Regularly maintain indexes by rebuilding or reorganizing them to ensure they remain efficient.

4. **Optimize Joins**

Join Order: Ensure the most restrictive join (the one that reduces the number of rows the most) is performed first.

Index Foreign Keys: Ensure that foreign keys used in joins are indexed.

Avoid Cartesian Joins: Make sure you have appropriate join conditions to avoid Cartesian products.

5. **Use Query Optimizations**

Limit Rows Returned: Use LIMIT (in MySQL, PostgreSQL) or TOP (in SQL Server) to limit the number of rows returned if possible.

Subqueries and Derived Tables: Optimize subqueries and derived tables to avoid unnecessary computations.

Union vs. Union All: Use UNION ALL if you do not need to eliminate duplicates, as it is faster than UNION.

6. **Partitioning**

Table Partitioning: Consider partitioning large tables based on range, list, or hash to improve query performance on large datasets.

Indexed Views: In some databases, indexed views can help to speed up frequently executed queries.

7. **Hardware and Configuration**

Hardware Resources: Ensure that the database server has adequate CPU, memory, and disk I/O capabilities.

Database Configuration: Tune database configuration parameters like buffer pool size, cache size, and maximum connections.