NAME: YAHYA A. G. ALSAEDI **TIME: 5:45 Hours**
GITHUB: YahyaAdnan

OPTIMZATION OF QUERY.

1. Problem Identification

The original query had multiple inefficiencies that can cause performance problems, including:

- choosing all columns (Jobs*, JobCategories*, etc.) which generates pointless information and raises network overhead.
- Using multiple **LEFT JOIN**s with filter will probably be duplicate.
- Using a lot of **LIKE** connotation will be expense due to lack of indexing.
- Pagination with **LIMIT** & **OFFSET** is not ideal for big database.
- grouping information using **GROUP BY** without any aggregate purpose unneeded.

2. Optimization Changes

I. Minimized SELECT Columns

- **Change Made:** Limited the SELECT list to only the columns essential for the query's purpose (e.g., Jobs.id, Jobs.name, JobCategories.name, JobTypes.name, etc.).
- **Why:** Reducing certain columns reduces memory consumption and data transmission, therefore accelerating query execution. Choosing empty columns squanders computing power as well as bandwidth.

II. Simplified JOIN Conditions

- **Change Made:** Eliminated useless **LEFT JOIN**s and keep only those necessary for display or filtering logic.
- **Why:**
  - **LEFT JOIN** increases the computational cost by keeping all rows from the left table independent of any match discovery. When feasible, switching to **INNER JOIN** helps to shrink the dataset early in query running.
  - Further accelerating join operations is ensuring indices on join columns (Jobs.id, JobCategories.id, JobTypes.id, etc.).

III. Removed Redundant GROUP BY

- **Change Made:** Removed GROUP BY Jobs.id unless aggregation functions like COUNT or SUM were required.
- **Why:**
    - If there's no aggregation (e.g., COUNT, MAX), GROUP BY is redundant. Removing it avoids unnecessary sorting and grouping operations in the query execution plan.

IV. Consolidated LIKE Conditions

- **Change Made:** Reduced several LIKE clauses into fewer tests on more relevant columns. As a result:
    - The query checks **Jobs.name, JobCategories.name, JobTypes.name**, and P**ersonalities.name**, which are likely the most important fields for searching.
- **Why:**
    - **LIKE '%value%'** scans the entire column for matches, which is slow on large datasets unless indexed.
    - Using fewer conditions reduces the CPU workload. Consideration was also given to using a full-text index or search for optimized text matching.

V. Improved Pagination

- **Change Made:** suggested pagination using indexed cursors instead of **LIMIT** and **OFFSET**
- **Why:**
    - **LIMIT** with **OFFSET** becomes slower as the **OFFSET** value increases because the database must scan and skip the rows. Using an indexed cursor allows the query to directly fetch the next set of rows.

VI. Removed Unnecessary Parentheses

- **Change Made:** Eliminated unnecessary parenthesis in **WHERE** clauses and **ON** clauses to simplify the search.
- **Why:**
    - Too many parenthesis could confound query analyzers, so execution plans become more difficult to understand and debug.

VII. Removed Unnecessary Parentheses

- **Change Made:** Replaced multiple **LIKE** '%**value**%' conditions with a single **MATCH**() clause using full-text search indexes on relevant columns such as **Jobs.name, Jobs.description, JobCategories.name, and JobTypes.name.**
- **Why:** Because **LIKE '%value%'** cannot utilize normal indexes, which makes it ineffective for big datasets, it triggers a complete table search. For this kind of search, full-text search using **MATCH**() is best and uses specialized full-text indexes to greatly minimize CPU and disk I/O. **MATCH**() also offers relevance score, which helps to better rank results depending on their degree of match to the search word.

3. Expected Improvements

Optimization	**Impact**
Minimized SELECT columns	Reduced data transfer and memory usage.
Simplified JOINs	Faster join operations and execution.
Consolidated LIKE filters	Reduced CPU workload on text matching.
Removed GROUP BY	Avoided unnecessary grouping overhead.
Optimized Pagination	Efficient handling of large datasets.
Added Indexes	Faster lookups for join/filter columns.

For both current and potential use scenarios, this optimized query strikes a mix of scalability, velocity, and readability. Not only may pagination be improved but also major speed improvements can be obtained by lowering pointless calculations by using indices.