Database Query Optimization

Task

Based on the given SQL query we need to optimize this for better performance.

Original Query

```
SELECT
c.Name,f.Role,
SUM(c.hours) AS Total_Tracked_Hours
SUM(f.Estimed Hours) AS Total_Allocated_Hours,
Date
FROM
ClickUp c
JOIN
Float f on c.Name = f.Name
GROUP BY
c.Name, f.Role
HAVING
SUM(c.hours) > 100
ORDER BY
Total_Allocated_Hours DESC;
```

Query Optimization Process

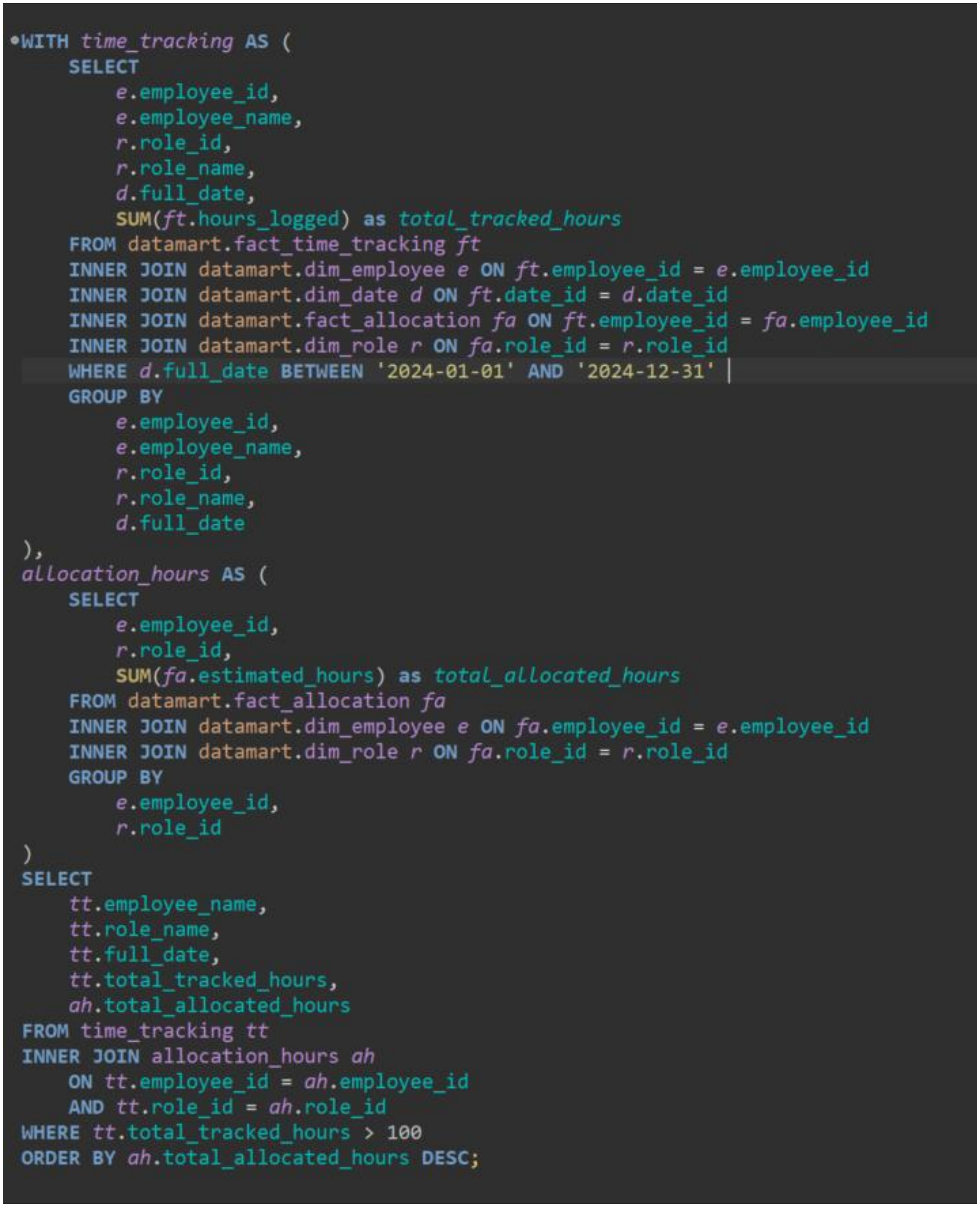
From the above query, we can observe the following issue and where to improve.

Origins Query Issues:

- *Absence of appropriate table joins and aliases*
- *Uncertain date column reference*
- *Inadequate join conditions*
- *No indexes were mentioned.*
- *Ineffective grouping*

Optimized Query

Using multiple CTEs to improve the readability of queries for easy understanding and better. For larger data, we added the filter where the date range in case of larger data. Note this is an option depending on the range of data you want to see.



```sql
WITH time_tracking AS (
    SELECT
        e.employee_id,
        e.employee_name,
        r.role_id,
        r.role_name,
        d.full_date,
        SUM(ft.hours_logged) as total_tracked_hours
    FROM datamart.fact_time_tracking ft
    INNER JOIN datamart.dim_employee e ON ft.employee_id = e.employee_id
    INNER JOIN datamart.dim_date d ON ft.date_id = d.date_id
    INNER JOIN datamart.fact_allocation fa ON ft.employee_id = fa.employee_id
    INNER JOIN datamart.dim_role r ON fa.role_id = r.role_id
    WHERE d.full_date BETWEEN '2024-01-01' AND '2024-12-31'
    GROUP BY
        e.employee_id,
        e.employee_name,
        r.role_id,
        r.role_name,
        d.full_date
),
allocation_hours AS (
    SELECT
        e.employee_id,
        r.role_id,
        SUM(fa.estimated_hours) as total_allocated_hours
    FROM datamart.fact_allocation fa
    INNER JOIN datamart.dim_employee e ON fa.employee_id = e.employee_id
    INNER JOIN datamart.dim_role r ON fa.role_id = r.role_id
    GROUP BY
        e.employee_id,
        r.role_id
)
SELECT
    tt.employee_name,
    tt.role_name,
    tt.full_date,
    tt.total_tracked_hours,
    ah.total_allocated_hours
FROM time_tracking tt
INNER JOIN allocation_hours ah
    ON tt.employee_id = ah.employee_id
    AND tt.role_id = ah.role_id
WHERE tt.total_tracked_hours > 100
ORDER BY ah.total_allocated_hours DESC;
```

Breakdown

Common Table Expressions (CTEs)

The query uses two Common Table Expressions (CTEs) to organize data before the final selection:

1. **First CTE (time_tracking)**

- Calculates total tracked hours per employee, role, and date.
- Filters data for the year 2024.
- Joins tables: ***fact_time_tracking, dim_employee, dim_date, fact_allocation, and dim_role***.

2. **Second CTE (allocation_hours)**

- Calculates total allocated hours per employee and role.
- Joins tables: ***fact_allocation, dim_employee, and dim_role***.

3. **Final Query**

- Combines data from the two CTEs using ***employee_id and role_id.***
- Filters for employees with tracked hours > 100.
- Outputs employee name, role name, date, tracked hours, and allocated hours.
- Sorts results by allocated hours in descending order.

Improve Table Structure by Indexing

To improve query performance by adding indexes on frequently joined and sorted columns.

Breakdown

Index Creation Statements

1. **Index on fact_time_tracking**

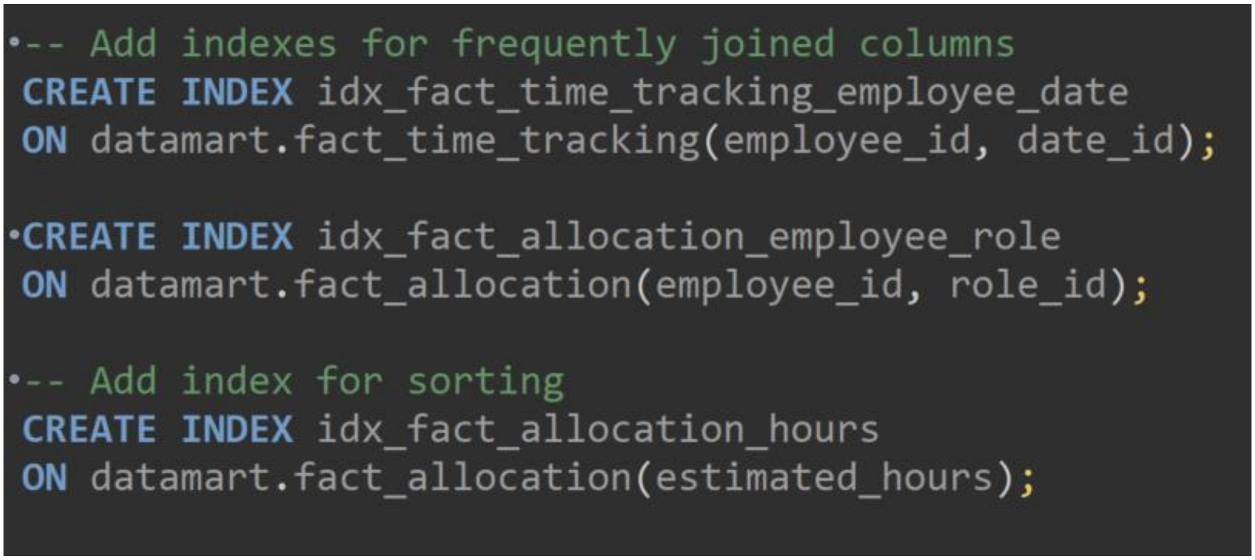
- Name: idx_fact_time_tracking_employee_date
- Columns: employee_id, date_id

2. **Index on fact_allocation**

- Name: idx_fact_allocation_employee_role
- Columns: employee_id, role_id

3. **Index for Sorting on fact_allocation**

- Name: idx_fact_allocation_hours
- Column: estimated_hours



```sql
-- Add indexes for frequently joined columns
CREATE INDEX idx_fact_time_tracking_employee_date
ON datamart.fact_time_tracking(employee_id, date_id);

CREATE INDEX idx_fact_allocation_employee_role
ON datamart.fact_allocation(employee_id, role_id);

-- Add index for sorting
CREATE INDEX idx_fact_allocation_hours
ON datamart.fact_allocation(estimated_hours);
```