🔍 Complex SQL Reporting — Employee Salary Insights

💡 FROM MULTI-CTES TO OPTIMIZED QUERIES

📌 How I Solved a Real SQL Reporting Scenario

I worked with a dataset containing:

- employees: ID, Name, Join Date, Department
- salary_history: Salary changes, promotions over time

Objective:

Build a report using real-world metrics based on salary & promotions.

🔎 Key Business Questions

- What is the **latest salary** of each employee?
- How many **promotions** did they get?
- What is the **maximum salary hike %**?
- Did their salary **ever decrease**?
- What's the **average time between changes**?
- How to **rank employees** based on salary growth?

CTE-Based Query Approach

Tools Used: SQL Server
Technique: Multiple CTEs for modular logic
CTEs Created:

- latest_salary_cte
- promotions_cte
- salary_growth_cte
- avg_months_cte
- salary_decreased_cte
- salary_growth_rank_cte

...and more!

```sql
select * from employees;
```

employee_id	name	join_date	department
1	Alice	2018-06-15	IT
2	Bob	2019-02-10	Finance
3	Charlie	2017-09-20	HR
4	David	2020-01-05	IT
5	Eve	2016-07-30	Finance
6	Sumit	2016-06-30	Finance

```sql
select * from salary_history;
```

employee_id	change_date	salary	promotion
1	2018-06-15	50000.00	No
1	2019-08-20	55000.00	No
1	2021-02-10	70000.00	Yes
2	2019-02-10	48000.00	No
2	2020-05-15	52000.00	Yes
2	2023-01-25	68000.00	Yes
3	2017-09-20	60000.00	No
3	2019-12-10	65000.00	No
3	2022-06-30	72000.00	Yes
4	2020-01-05	45000.00	No
4	2021-07-18	49000.00	No
5	2016-07-30	55000.00	No
5	2018-11-22	62000.00	Yes
5	2021-09-10	75000.00	Yes
6	2016-06-30	55000.00	No
6	2017-11-22	50000.00	No
6	2018-11-22	40000.00	No
6	2021-09-10	75000.00	Yes

```sql
with cte as (
select *,
rank() over (partition by employee_id order by change_date desc) as rn_desc,
rank() over (partition by employee_id order by change_date asc) as rn_asc
from salary_history
)
, latest_salary_cte as(
select employee_id, salary as latest_salary
from cte
where rn_desc = 1
)
, promotions_cte as (
select employee_id, count(*) as no_of_promotions
from cte
where promotion = 'Yes'
group by employee_id
)
, prev_salary_cte as (
select *,
lead(salary, 1) over (partition by employee_id order by change_date desc) as prev_salary,
lead(change_date, 1) over (partition by employee_id order by change_date desc) as prev_change_date
from cte
)
, salary_growth_cte as (
select employee_id, max(cast((salary-prev_salary)*100.0/prev_salary AS decimal(4,2))) as salary_growth
from prev_salary_cte
group by employee_id
)
, salary_decreased_cte as (
select distinct employee_id, 'N' as never_decreased
from prev_salary_cte
where salary < prev_salary
)
```

```
, avg_months_cte as (
select employee_id, avg(DATEDIFF(MONTH, prev_change_date, change_date)) as avg_months_between_changes
from prev_salary_cte
group by employee_id
)
, salary_ratio_cte as (
select employee_id,
max(case when rn_desc = 1 then salary end) / max(case when rn_asc = 1 then salary end) as salary_growth_ratio,
min(change_date) as join_date
from cte
group by employee_id
)
, salary_growth_rank_cte as (
select employee_id,
rank() over (order by salary_growth_ratio desc, join_date asc) as RankByGrowth
from salary_ratio_cte
)
select e.employee_id, name, isnull(p.no_of_promotions, 0) as no_of_promotions, msg.salary_growth,
isnull(sd.never_decreased, 'Y') as never_decreased, am.avg_months_between_changes,rbg.RankByGrowth
from employees e
left join latest_salary_cte s on e.employee_id = s.employee_id
left join promotions_cte p on e.employee_id = p.employee_id
left join salary_growth_cte msg on e.employee_id = msg.employee_id
left join salary_decreased_cte sd on e.employee_id = sd.employee_id
left join avg_months_cte am on e.employee_id = am.employee_id
left join salary_growth_rank_cte rbg on e.employee_id = rbg.employee_id
```

⚙️ Final Output (CTE Version)

employee_id	name	no_of_promotions	salary_growth	never_decreased	avg_months_between_changes	RankByGrowth
1	Alice	1	27.27	Y	16	2
2	Bob	2	30.77	Y	23	1
3	Cha...	1	10.77	Y	28	5
4	David	0	8.89	Y	18	6
5	Eve	2	20.97	Y	31	4
6	Sumit	1	87.50	N	21	3

🚀 Optimization — No CTE Version

🎯 Challenge: Re-write same logic in **one query**, no CTEs
💡 Used:

- CASE statements
- LEAD() + RANK()
- Aggregates with GROUP BY
- Logical conditions for better performance

```sql
with cte as (
select *,
rank() over (partition by employee_id order by change_date desc) as rn_desc,
rank() over (partition by employee_id order by change_date asc) as rn_asc,
lead(salary, 1) over (partition by employee_id order by change_date desc) as prev_salary,
lead(change_date, 1) over (partition by employee_id order by change_date desc) as prev_change_date
from salary_history
)
, salary_ratio_cte as (
select employee_id,
max(case when rn_desc = 1 then salary end) / max(case when rn_asc = 1 then salary end) as salary_growth_ratio,
min(change_date) as join_date
from cte
group by employee_id
)
select cte.employee_id,
max(case when rn_desc = 1 then salary end) as latest_salary,
sum(case when promotion = 'Yes' then 1 else 0 end) as no_of_promotions,
max(cast((salary-prev_salary)*100.0/prev_salary AS decimal(4,2))) as salary_growth,
case when max(case when salary < prev_salary then 1 else 0 end) = 0 then 'Y' else 'N' end as NeverDecreased,
avg(DATEDIFF(MONTH, prev_change_date, change_date)) as avg_months_between_changes,
rank() over (order by sr.salary_growth_ratio desc, sr.join_date asc) as RankByGrowth
from cte
left join salary_ratio_cte sr on cte.employee_id = sr.employee_id
group by cte.employee_id, sr.salary_growth_ratio, sr.join_date
order by cte.employee_id;
```

📈 Optimization Results

Same Results, Better Query Structure!

- ✅ Reduced CTE layers
- ✅ Easier to maintain
- ✅ Improved readability
- ✅ Great for reports

employee_id	latest_salary	no_of_promotions	salary_growth	NeverDecreased	avg_months_between_changes	RankByGrowth
1	70000.00	1	27.27	Y	16	2
2	68000.00	2	30.77	Y	23	1
3	72000.00	1	10.77	Y	28	5
4	49000.00	0	8.89	Y	18	6
5	75000.00	2	20.97	Y	31	4
6	75000.00	1	87.50	N	21	3