Database Query Optimization

1) Existing Query:

```
EXPLAIN ANALYZE
SELECT
        t.employee_id,
        (SELECT name FROM employee e WHERE e.id = t.employee_id),
        (SELECT name FROM project p WHERE p.id = t.project_id),
 SUM(EXTRACT(EPOCH FROM (t.time_to - t.time_from)) / 3600)
 FROM time_record t
 WHERE t.time_from >= NOW() - INTERVAL '1 month'
 GROUP BY
        t.employee_id,
        (SELECT name FROM employee e WHERE e.id = t.employee_id),
        (SELECT name FROM project p WHERE p.id = t.project_id)
ORDER BY
        (SELECT name FROM employee e WHERE e.id = t.employee_id),
        (SELECT name FROM project p WHERE p.id = t.project_id);
```

EXPLAIN ANALYZE result:

```
"GroupAggregate  (cost=7450.14..10514.78 rows=187 width=592) (actual
time=0.400..0.403 rows=2 loops=1)"
"  Group Key: ((SubPlan 1)), ((SubPlan 2)), t.employee_id"
"  ->  Sort  (cost=7450.14..7451.27 rows=453 width=576) (actual time=0.372..0.373
rows=3 loops=1)"
"        Sort Key: ((SubPlan 1)), ((SubPlan 2)), t.employee_id"
"        Sort Method: quicksort  Memory: 25kB"
"        ->  Seq Scan on time_record t  (cost=0.00..7430.16 rows=453 width=576) (actual
time=0.305..0.313 rows=3 loops=1)"
"              Filter: (time_from >= (now() - '1 mon'::interval))"
"              SubPlan 1"
"                ->  Index Scan using employee_pkey on employee e  (cost=0.15..8.17
rows=1 width=138) (actual time=0.028..0.028 rows=1 loops=3)"
"                      Index Cond: (id = t.employee_id)"
"              SubPlan 2"
"                ->  Index Scan using project_pkey on project p  (cost=0.14..8.16 rows=1
width=418) (actual time=0.021..0.021 rows=1 loops=3)"
"                      Index Cond: (id = t.project_id)"
"Planning Time: 0.755 ms"
"Execution Time: 0.629 ms"
```

Identified bottleneck:

- No Join were used in the existing query
- Found there were 2 SubPlans and loops 3 times as per 3 records row to each employee and project in the existing query. If the dataset is larger, then the subquery will loop many times

2) Optimized Query:

```
EXPLAIN ANALYZE
SELECT t.employee_id,
                e.name,
                p.name,
                SUM(EXTRACT(EPOCH FROM (t.time_to - t.time_from)) / 3600)
total_hours
FROM time_record t
INNER JOIN employee e ON e.id = t.employee_id
INNER JOIN project p on p.id = t.project_id
WHERE t.time_from >= NOW() - INTERVAL '1 month'
GROUP BY t.employee_id, e.name, p.name
ORDER BY e.name, p.name;
```

EXPLAIN ANALYZE result:

```
"QUERY PLAN"
"GroupAggregate  (cost=86.74..99.25 rows=385 width=592) (actual time=0.170..0.178
rows=2 loops=1)"
"  Group Key: e.name, p.name, t.employee_id"
"  ->  Sort  (cost=86.74..87.70 rows=385 width=576) (actual time=0.123..0.125 rows=3
loops=1)"
"        Sort Key: e.name, p.name, t.employee_id"
"        Sort Method: quicksort  Memory: 25kB"
"        ->  Hash Join  (cost=34.18..70.21 rows=385 width=576) (actual time=0.104..0.109
rows=3 loops=1)"
"              Hash Cond: (t.employee_id = e.id)"
"              ->  Hash Join  (cost=13.82..48.84 rows=385 width=438) (actual
time=0.072..0.076 rows=3 loops=1)"
"                    Hash Cond: (t.project_id = p.id)"
"                    ->  Seq Scan on time_record t  (cost=0.00..33.80 rows=453 width=24)
(actual time=0.024..0.026 rows=3 loops=1)"
"                          Filter: (time_from >= (now() - '1 mon'::interval))"
```

"   -> Hash (cost=11.70..11.70 rows=170 width=426) (actual time=0.035..0.035 rows=2 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 9kB"
"   -> Seq Scan on project p (cost=0.00..11.70 rows=170 width=426) (actual time=0.005..0.006 rows=2 loops=1)"
"   -> Hash (cost=14.60..14.60 rows=460 width=146) (actual time=0.005..0.006 rows=2 loops=1)"
"   Buckets: 1024 Batches: 1 Memory Usage: 9kB"
"   -> Seq Scan on employee e (cost=0.00..14.60 rows=460 width=146) (actual time=0.004..0.004 rows=2 loops=1)"
"Planning Time: 0.619 ms"
"Execution Time: 0.311 ms"

After optimization:

- Uses INNER JOIN, so it uses hash join and scan only once on each project and employee table
- The query no longer use SubPlan
- The cost reduce from 7450.14..10514.78 into 86.74..99.25

Indexing strategies:

- Create index on time_from column in the time_record field, to speed up the interval filter
- Create index on employee_id and project_id in the time_record table for join