# INFORME TECNICO

## Introduccion

El presente informe técnico se enfoca en analizar y comparar estrategias de optimización de consultas en SQL, centrándose en el uso de SQL básico, FAV (window functions), y la cláusula WITH (common table expressions). Se han planteado diversas problemáticas basadas en un caso de estudio (Biblioteca Nacional), evaluando los tiempos de ejecución de cada enfoque para determinar la opción más eficiente.

## Desarrollo

### FAV - WITH Benchmark

Se crearon 5 problematicas del caso de estudio (Biblioteca Nacional) y de cada uno se mostrara una posible implementacion con sql basico, fav, y la clausula with, en conjunto con los tiempos medidos

1. Crear 5 problematicas del caso de estudio
    1. aplicar sql basico, fav, with

2. explain analyse (screenshot)
3. escojer el mejor

ejemplo 1

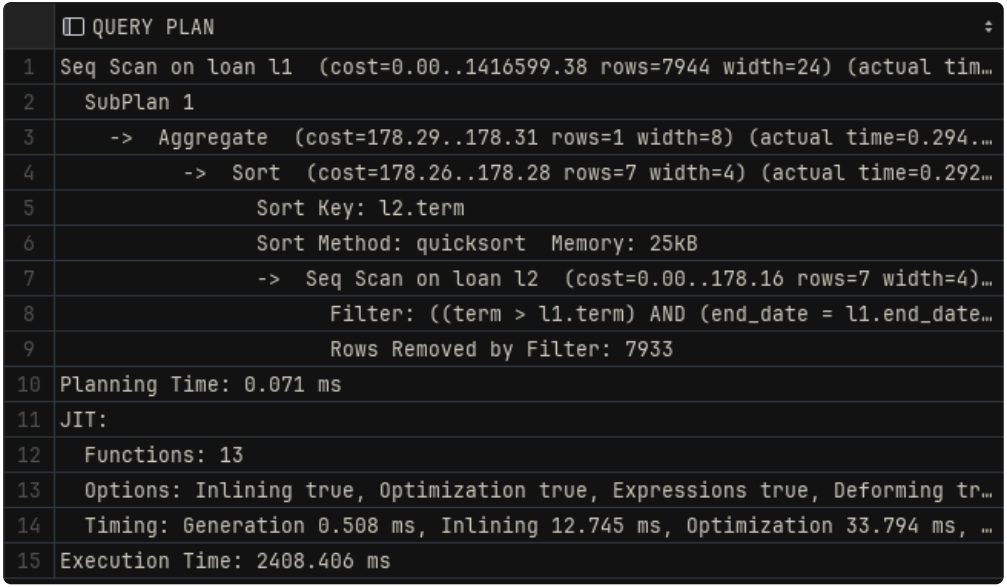
Se necesita de los prestamos que se entregan en un dia darle prioridad a los que han estado mas tiempo prestados, para eso:

Se desea mostrar los datos de los prestamos (id_document, id_service, fecha inicio, fecha entrega), asi como la prioridad de los prestamos que se entregan en un mismo dia con 1 la mas alta que tendra el que mas tiempo a estado prestado (pueden haber empates)

**SQL Basico**

```
select id_document,
       id_service,
       start_date,
       end_date,
       (select count(distinct term) + 1
        from loan l2
        where l2.end_date = l1.end_date
```

```
           and l2.term > l1.term) as priority
from loan l1;
```

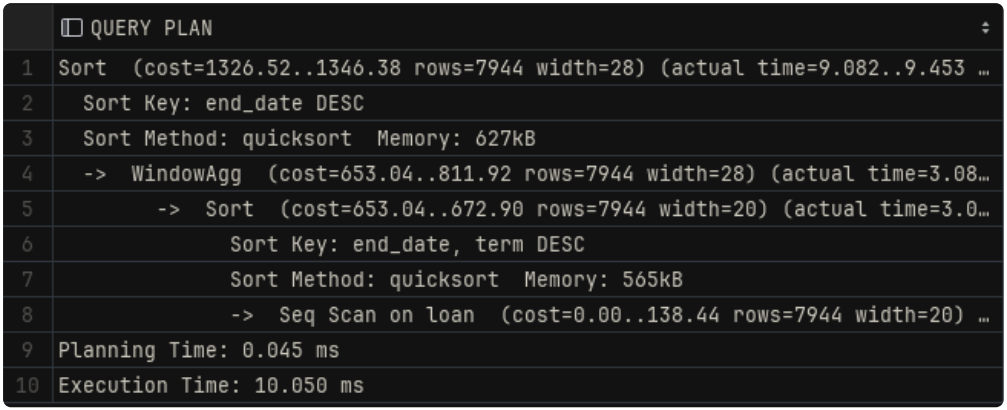


```
QUERY PLAN
Seq Scan on loan l1  (cost=0.00..1416599.38 rows=7944 width=24) (actual tim…
  SubPlan 1
    ->  Aggregate  (cost=178.29..178.31 rows=1 width=8) (actual time=0.294.…
          ->  Sort  (cost=178.26..178.28 rows=7 width=4) (actual time=0.292…
                Sort Key: l2.term
                Sort Method: quicksort  Memory: 25kB
                ->  Seq Scan on loan l2  (cost=0.00..178.16 rows=7 width=4)…
                      Filter: ((term > l1.term) AND (end_date = l1.end_date…
                      Rows Removed by Filter: 7933
Planning Time: 0.071 ms
JIT:
  Functions: 13
  Options: Inlining true, Optimization true, Expressions true, Deforming tr…
  Timing: Generation 0.508 ms, Inlining 12.745 ms, Optimization 33.794 ms, …
Execution Time: 2408.406 ms
```

## FAV

```
select loan.id_document,
       loan.id_service,
       loan.start_date,
```

```
       loan.end_date,
       rank() over (partition by end_date order by term desc) as priority
from loan
order by end_date desc;
```

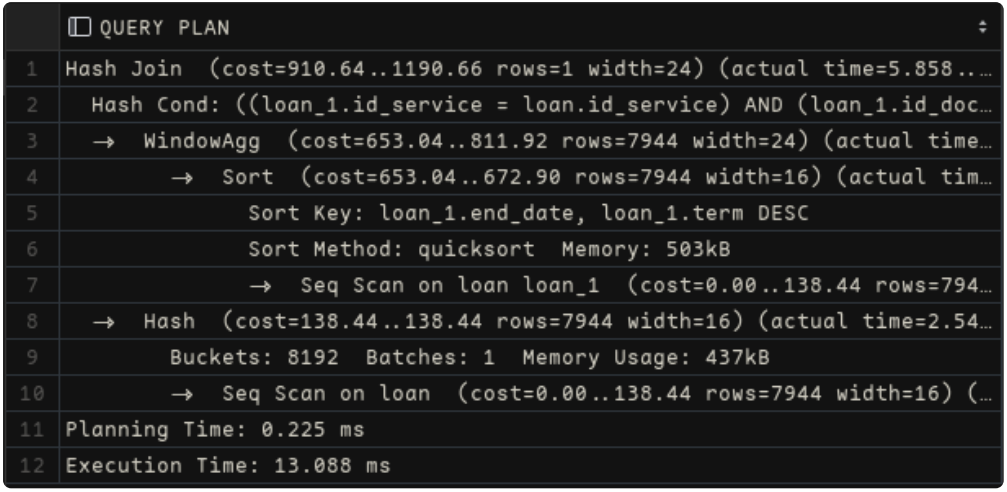


| | QUERY PLAN |
|---|---|
| 1 | Sort  (cost=1326.52..1346.38 rows=7944 width=28) (actual time=9.082..9.453 … |
| 2 | Sort Key: end_date DESC |
| 3 | Sort Method: quicksort  Memory: 627kB |
| 4 | ->  WindowAgg  (cost=653.04..811.92 rows=7944 width=28) (actual time=3.08… |
| 5 | ->  Sort  (cost=653.04..672.90 rows=7944 width=20) (actual time=3.0… |
| 6 | Sort Key: end_date, term DESC |
| 7 | Sort Method: quicksort  Memory: 565kB |
| 8 | ->  Seq Scan on loan  (cost=0.00..138.44 rows=7944 width=20) … |
| 9 | Planning Time: 0.045 ms |
| 10 | Execution Time: 10.050 ms |

## WITH

```
explain analyze
with ranking as (select id_service, id_document,
                        rank() over (partition by end_date order by term desc) as
priority
                 from loan)
```

```sql
select loan.id_document, loan.id_service, start_date, end_date, priority
from loan
         join ranking on loan.id_service = ranking.id_service
    and loan.id_document = ranking.id_document;
```



```
QUERY PLAN
Hash Join  (cost=910.64..1190.66 rows=1 width=24) (actual time=5.858..…
  Hash Cond: ((loan_1.id_service = loan.id_service) AND (loan_1.id_doc…
  →  WindowAgg  (cost=653.04..811.92 rows=7944 width=24) (actual time…
        →  Sort  (cost=653.04..672.90 rows=7944 width=16) (actual tim…
              Sort Key: loan_1.end_date, loan_1.term DESC
              Sort Method: quicksort  Memory: 503kB
              →  Seq Scan on loan loan_1  (cost=0.00..138.44 rows=794…
  →  Hash  (cost=138.44..138.44 rows=7944 width=16) (actual time=2.54…
        Buckets: 8192  Batches: 1  Memory Usage: 437kB
        →  Seq Scan on loan  (cost=0.00..138.44 rows=7944 width=16) (…
Planning Time: 0.225 ms
Execution Time: 13.088 ms
```

Resultados: FAV esa la mejor opcion por simplicidad y velocidad

ejemplo 2

Se quiere hacer un censo de los miembros que son estudiantes universitarios jovenes asociados a la biblioteca asi como el promedio de edad por pais de los estudiantes

De ellos se desea obtener los datos (nombre, edad, pais, escuela) asi como el promedio de edad

**SQL Basico**

```
select m1.id_member,
       name,
       age,
       country,
       school,
       (select avg(age)
        from member m2
        where m1.country = m2.country
          and m1.category = 'student') as average_age
from member m1
         join student on m1.id_member = student.id_member
where age between 20 and 40
  and school like 'University%';
```

| | QUERY PLAN |
|---|---|
| 1 | Hash Join  (cost=1059.97..13008552.64 rows=3529 width=99) (actual time=237.… |
| 2 |   Hash Cond: (m1.id_member = student.id_member) |
| 3 |   ->  Seq Scan on member m1  (cost=0.00..4109.15 rows=42858 width=39) (actu… |
| 4 |         Filter: ((age >= 20) AND (age <= 40)) |
| 5 |         Rows Removed by Filter: 126988 |
| 6 |   ->  Hash  (cost=884.50..884.50 rows=14038 width=36) (actual time=6.425..6… |
| 7 |         Buckets: 16384  Batches: 1  Memory Usage: 1064kB |
| 8 |         ->  Seq Scan on student  (cost=0.00..884.50 rows=14038 width=36) (a… |
| 9 |               Filter: ((school)::text ~~ 'University%'::text) |
| 10 |               Rows Removed by Filter: 28388 |
| 11 |   SubPlan 1 |
| 12 |     ->  Aggregate  (cost=3684.68..3684.69 rows=1 width=32) (actual time=17.… |
| 13 |           ->  Result  (cost=0.00..3682.97 rows=685 width=4) (actual time=0.… |
| 14 |                 One-Time Filter: (m1.category = 'student'::category_type) |
| 15 |                 ->  Seq Scan on member m2  (cost=0.00..3682.97 rows=685 wid… |
| 16 |                       Filter: ((m1.country)::text = (country)::text) |
| 17 |                       Rows Removed by Filter: 127700 |
| 18 | Planning Time: 0.352 ms |
| 19 | JIT: |
| 20 |   Functions: 26 |
| 21 |   Options: Inlining true, Optimization true, Expressions true, Deforming tr… |
| 22 |   Timing: Generation 0.994 ms, Inlining 21.128 ms, Optimization 111.775 ms,… |
| 23 | Execution Time: 63055.212 ms |

## FAV

```sql
select member.id_member,
       name,
       age,
       country,
       school,
       avg(age) over (partition by country) as average_age
from member
         join student on member.id_member = student.id_member
where age between 20 and 40
  and school like 'University%';
```

| | QUERY PLAN |
|---|---|
| 1 | WindowAgg  (cost=5489.58..5551.34 rows=3529 width=99) (actual time=28.615..… |
| 2 | ->  Sort  (cost=5489.58..5498.40 rows=3529 width=67) (actual time=28.591.… |
| 3 | Sort Key: member.country |
| 4 | Sort Method: quicksort  Memory: 446kB |
| 5 | ->  Hash Join  (cost=1059.97..5281.63 rows=3529 width=67) (actual t… |
| 6 | Hash Cond: (member.id_member = student.id_member) |
| 7 | ->  Seq Scan on member  (cost=0.00..4109.15 rows=42858 width=… |
| 8 | Filter: ((age >= 20) AND (age <= 40)) |
| 9 | Rows Removed by Filter: 126988 |
| 10 | ->  Hash  (cost=884.50..884.50 rows=14038 width=36) (actual t… |
| 11 | Buckets: 16384  Batches: 1  Memory Usage: 1064kB |
| 12 | ->  Seq Scan on student  (cost=0.00..884.50 rows=14038 … |
| 13 | Filter: ((school)::text ~~ 'University%'::text) |
| 14 | Rows Removed by Filter: 28388 |
| 15 | Planning Time: 0.275 ms |
| 16 | Execution Time: 30.826 ms |

## WITH

```
with average_age_per_country as (select country, avg(age) as average_age
                                 from member
                                 group by country)
select member.id_member,
       name,
       age,
```

```
       member.country,
       school,
       average_age
from member
         join student on member.id_member = student.id_member
         join average_age_per_country on member.country =
average_age_per_country.country
where school like 'University%';
```

| | QUERY PLAN |
|---|---|
| 1 | Hash Join  (cost=5167.19..8908.95 rows=14038 width=99) (actual time=41.680… |
| 2 | Hash Cond: ((member.country)::text = (average_age_per_country.country)::… |
| 3 | ->  Hash Join  (cost=1059.97..4764.26 rows=14038 width=67) (actual time=… |
| 4 | Hash Cond: (member.id_member = student.id_member) |
| 5 | ->  Seq Scan on member  (cost=0.00..3256.77 rows=170477 width=35) … |
| 6 | ->  Hash  (cost=884.50..884.50 rows=14038 width=36) (actual time=6… |
| 7 | Buckets: 16384  Batches: 1  Memory Usage: 1064kB |
| 8 | ->  Seq Scan on student  (cost=0.00..884.50 rows=14038 width… |
| 9 | Filter: ((school)::text ~~ 'University%'::text) |
| 10 | Rows Removed by Filter: 28388 |
| 11 | ->  Hash  (cost=4104.10..4104.10 rows=249 width=40) (actual time=35.432.… |
| 12 | Buckets: 1024  Batches: 1  Memory Usage: 22kB |
| 13 | ->  Subquery Scan on average_age_per_country  (cost=4068.62..4104.… |
| 14 | ->  Finalize GroupAggregate  (cost=4068.62..4101.61 rows=249… |
| 15 | Group Key: member_1.country |
| 16 | ->  Gather Merge  (cost=4068.62..4097.26 rows=249 widt… |
| 17 | Workers Planned: 1 |
| 18 | Workers Launched: 1 |
| 19 | ->  Sort  (cost=3068.61..3069.23 rows=249 width=… |
| 20 | Sort Key: member_1.country |
| 21 | Sort Method: quicksort  Memory: 43kB |
| 22 | Worker 0:  Sort Method: quicksort  Memory:… |
| 23 | ->  Partial HashAggregate  (cost=3056.21..… |
| 24 | Group Key: member_1.country |
| 25 | Batches: 1  Memory Usage: 93kB |
| 26 | Worker 0:  Batches: 1  Memory Usage:… |
| 27 | ->  Parallel Seq Scan on member memb… |
| 28 | Planning Time: 0.321 ms |
| 29 | Execution Time: 74.956 ms |

Resultados: FAV esa la mejor opcion por simplicidad y velocidad

ejemplo 3

Se desea conocer el titulo, fecha de creacion, formato, genero y promedio de duracion de las ultimas 5 canciones anadidas a su catalogo, y la duracion
total segun el genero al que pertenezcan

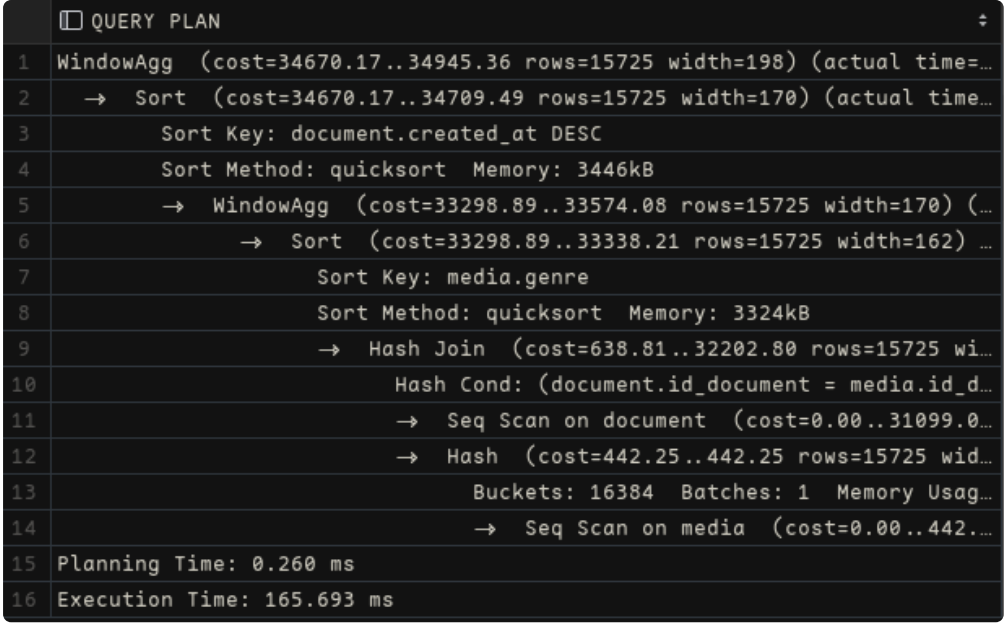
**SQL Basico**

```
select title,
       created_at,
       format,
       genre,
       (select avg(last_five_documents.duration)
        from (select duration
              from document d2
                       join media m2 on d2.id_document = m2.id_document
              where d2.created_at <= d1.created_at
              order by d2.created_at desc
              limit 5) as last_five_documents) as average_duration,
       (select sum(m2.duration)
```

```
        from document d2
                 join media m2 on d2.id_document = m2.id_document
        where m2.genre = m1.genre)             as total_per_genre
from document d1
         join media m1 on d1.id_document = m1.id_document;
```

### FAV

```
select title,
       created_at,
       format,
       genre,
       avg(duration) over (order by created_at desc rows between 5 preceding and
current row ) as average_duration,
       sum(duration) over (partition by genre)
as average_duration_per_genre
from document
         join media on document.id_document = media.id_document;
```




## WITH

```
with average_duration as (select document.id_document,
                                 genre,
                                 avg(duration) over (order by created_at desc
rows 5 preceding) as average
                          from document
                                   join media on document.id_document =
```

```
media.id_document),
     total_duration_per_genre as (select genre, sum(duration) as total_duration
                                  from media
                                  group by genre)
select title,
       created_at,
       format,
       ad.genre,
       average,
       total_duration
from document d
         join average_duration ad on d.id_document = ad.id_document
         join total_duration_per_genre td on ad.genre = td.genre;
```

```
QUERY PLAN
Hash Join  (cost=34959.83..66921.59 rows=15725 width=198) (actual time=…
  Hash Cond: ((ad.genre)::text = (td.genre)::text)
  →  Hash Join  (cost=33927.89..65848.36 rows=15725 width=190) (actual…
        Hash Cond: (d.id_document = ad.id_document)
        →  Seq Scan on document d  (cost=0.00..31099.07 rows=177107 wi…
        →  Hash  (cost=33731.33..33731.33 rows=15725 width=111) (actua…
              Buckets: 16384  Batches: 1  Memory Usage: 2034kB
              →  Subquery Scan on ad  (cost=33298.89..33731.33 rows=15…
                    →  WindowAgg  (cost=33298.89..33574.08 rows=15725 …
                          →  Sort  (cost=33298.89..33338.21 rows=15725…
                                Sort Key: document.created_at DESC
                                Sort Method: quicksort  Memory: 2157kB
                                →  Hash Join  (cost=638.81..32202.80 r…
                                      Hash Cond: (document.id_document …
                                      →  Seq Scan on document  (cost=0…
                                      →  Hash  (cost=442.25..442.25 ro…
                                            Buckets: 16384  Batches: 1 …
                                            →  Seq Scan on media  (cos…
  →  Hash  (cost=835.38..835.38 rows=15725 width=83) (actual time=14.6…
        Buckets: 16384  Batches: 1  Memory Usage: 1909kB
        →  Subquery Scan on td  (cost=520.88..835.38 rows=15725 width=…
              →  HashAggregate  (cost=520.88..678.12 rows=15725 width=…
                    Group Key: media_1.genre
                    Batches: 1  Memory Usage: 3345kB
                    →  Seq Scan on media media_1  (cost=0.00..442.25 r…
Planning Time: 0.410 ms
Execution Time: 235.391 ms
```

Resultados: FAV esa la mejor opcion por simplicidad y velocidad

ejemplo 4

Se desea conocer los datos de profesionales que tienen deudas y el total de deuda relacionada a empleados que trabajan para la misma empresa

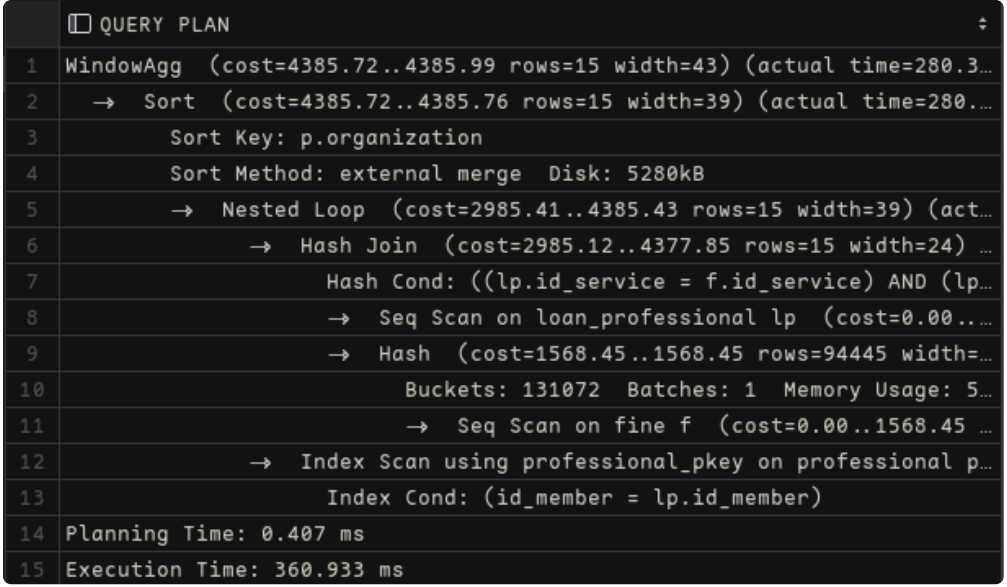
**SQL basico**

```
select f.id_fine,
       f.id_document,
       f.id_service,
       p.organization,
       (select sum(fee)
        from fine f2
                 join loan_professional lp2 on f2.id_service = lp2.id_service
            and f2.id_document = lp2.id_document
                 join professional p2 on lp2.id_member = p2.id_member
            and f2.id_document = lp2.id_document
        where p2.organization = p.organization) as total_fee
from fine f
         join loan_professional lp on f.id_service = lp.id_service
```

```
    and f.id_document = lp.id_document
         join professional p on lp.id_member = p.id_member
    and f.id_document = lp.id_document;
```

**FAV**

```
select id_fine,
       f.id_document,
       f.id_service,
       p.organization,
       sum(fee) over (partition by p.organization) as total_fee
from fine f
         join loan_professional lp on f.id_service = lp.id_service
    and f.id_document = lp.id_document
         join professional p on lp.id_member = p.id_member
    and f.id_document = lp.id_document;
```

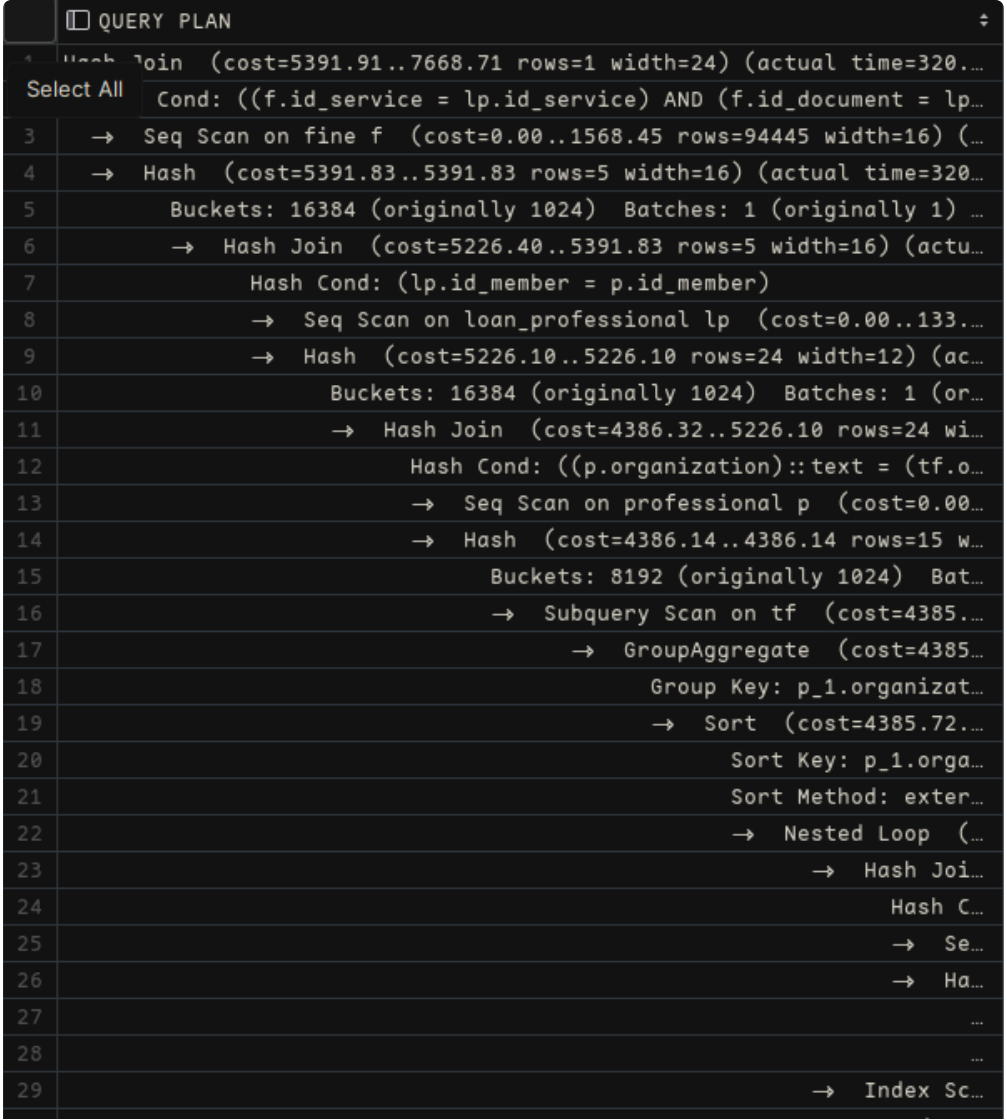


```
QUERY PLAN
WindowAgg  (cost=4385.72..4385.99 rows=15 width=43) (actual time=280.3…
  →  Sort  (cost=4385.72..4385.76 rows=15 width=39) (actual time=280.…
        Sort Key: p.organization
        Sort Method: external merge  Disk: 5280kB
        →  Nested Loop  (cost=2985.41..4385.43 rows=15 width=39) (act…
              →  Hash Join  (cost=2985.12..4377.85 rows=15 width=24) …
                    Hash Cond: ((lp.id_service = f.id_service) AND (lp…
                    →  Seq Scan on loan_professional lp  (cost=0.00..…
                    →  Hash  (cost=1568.45..1568.45 rows=94445 width=…
                          Buckets: 131072  Batches: 1  Memory Usage: 5…
                          →  Seq Scan on fine f  (cost=0.00..1568.45 …
              →  Index Scan using professional_pkey on professional p…
                    Index Cond: (id_member = lp.id_member)
Planning Time: 0.407 ms
Execution Time: 360.933 ms
```

## WITH

```
with total_fee_per_organization as (select organization, sum(fee) as total
                                    from fine f
                                             join loan_professional lp on
f.id_service = lp.id_service
                                                 and f.id_document = lp.id_document
                                             join professional p on lp.id_member
= p.id_member
```

```
                                        and f.id_document = lp.id_document
                                    group by organization)
select f.id_fine, f.id_document, f.id_service, tf.total
from fine f
         join loan_professional lp on f.id_service = lp.id_service
    and f.id_document = lp.id_document
         join professional p on lp.id_member = p.id_member
    and f.id_document = lp.id_document
         join total_fee_per_organization tf on p.organization = tf.organization;
```


QUERY PLAN
Select All
Cond: ((f.id_service = lp.id_service) AND (f.id_document = lp…
→ Seq Scan on fine f (cost=0.00..1568.45 rows=94445 width=16) (…
→ Hash (cost=5391.83..5391.83 rows=5 width=16) (actual time=320…
Buckets: 16384 (originally 1024) Batches: 1 (originally 1) …
→ Hash Join (cost=5226.40..5391.83 rows=5 width=16) (actu…
Hash Cond: (lp.id_member = p.id_member)
→ Seq Scan on loan_professional lp (cost=0.00..133.…
→ Hash (cost=5226.10..5226.10 rows=24 width=12) (ac…
Buckets: 16384 (originally 1024) Batches: 1 (or…
→ Hash Join (cost=4386.32..5226.10 rows=24 wi…
Hash Cond: ((p.organization)::text = (tf.o…
→ Seq Scan on professional p (cost=0.00…
→ Hash (cost=4386.14..4386.14 rows=15 w…
Buckets: 8192 (originally 1024) Bat…
→ Subquery Scan on tf (cost=4385.…
→ GroupAggregate (cost=4385…
Group Key: p_1.organizat…
→ Sort (cost=4385.72.…
Sort Key: p_1.orga…
Sort Method: exter…
→ Nested Loop (…
→ Hash Joi…
Hash C…
→ Se…
→ Ha…
…
…
→ Index Sc…

```
30                                                    Index …
31  Planning Time: 0.618 ms
32  Execution Time: 357.636 ms
```

Resultados: FAV y WITH estan casi empatados dado que la diferencia es demasiado pequena y las muestras fueron muy pocas

ejemplo 5

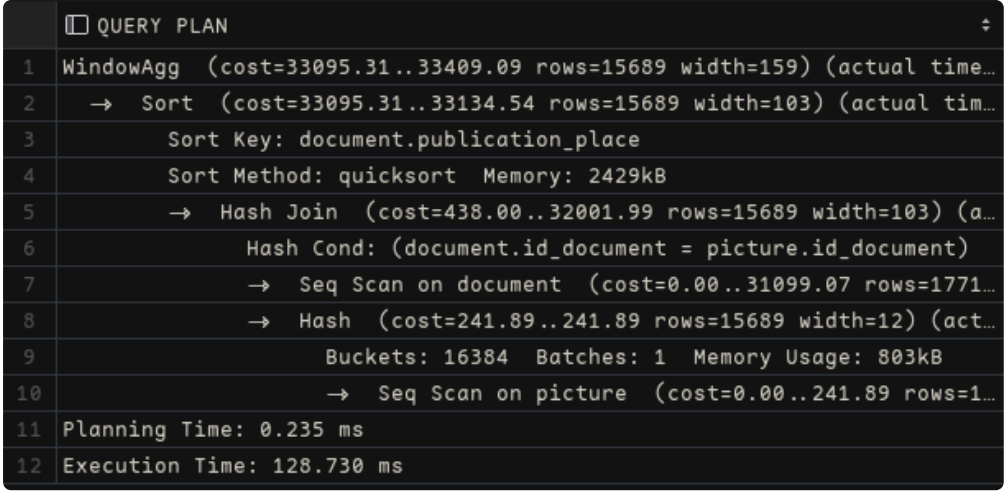
**SQL Basico**

```sql
select d.id_document,
       title,
       format,
       publication_place,
       (select avg(p2.dimension_height)
        from document d2
                 join picture p2 on d2.id_document = p2.id_document
        where d2.publication_place = d.publication_place) as average_height,
       (select avg(p2.dimension_width)
        from document d2
                 join picture p2 on d2.id_document = p2.id_document
        where d2.publication_place = d.publication_place) as average_width
```

```
from document d
         join picture p on d.id_document = p.id_document;
```

### FAV

```
select document.id_document,
       title,
       format,
       publication_place,
       avg(dimension_width) over (partition by publication_place)  as
average_width,
       avg(dimension_height) over (partition by publication_place) as
average_height
from document
         join picture on document.id_document = picture.id_document;
```

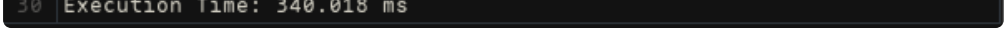



## WITH

```
with averate_height as (select publication_place, avg(dimension_height) as
avg_height
                        from document
                                 join picture on document.id_document =
picture.id_document
                        group by publication_place),
     averate_width as (select publication_place, avg(dimension_width) as
avg_width
                       from document
```

```
                                join picture on document.id_document =
picture.id_document
                        group by publication_place)
select document.id_document, title, format, averate_width.publication_place,
avg_height, avg_width
from document
         join averate_height on document.publication_place =
averate_height.publication_place
         join averate_width on document.publication_place =
averate_width.publication_place;
```

| | QUERY PLAN |
|---|---|
| 1 | Hash Join  (cost=65102.21..96815.48 rows=18251 width=159) (actual ti… |
| 2 |  Hash Cond: ((document.publication_place)::text = (document_2.publi… |
| 3 |  →  Hash Join  (cost=32629.55..64193.57 rows=56854 width=139) (act… |
| 4 |  Hash Cond: ((document.publication_place)::text = (averate_he… |
| 5 |  →  Seq Scan on document  (cost=0.00..31099.07 rows=177107 w… |
| 6 |  →  Hash  (cost=32433.44..32433.44 rows=15689 width=44) (act… |
| 7 |  Buckets: 16384  Batches: 1  Memory Usage: 750kB |
| 8 |  →  Subquery Scan on averate_height  (cost=32080.43..3… |
| 9 |  →  HashAggregate  (cost=32080.43..32276.55 rows… |
| 10 |  Group Key: document_1.publication_place |
| 11 |  Batches: 1  Memory Usage: 2577kB |
| 12 |  →  Hash Join  (cost=438.00..32001.99 rows… |
| 13 |  Hash Cond: (document_1.id_document =… |
| 14 |  →  Seq Scan on document document_1 … |
| 15 |  →  Hash  (cost=241.89..241.89 rows=… |
| 16 |  Buckets: 16384  Batches: 1  Me… |
| 17 |  →  Seq Scan on picture  (cost… |
| 18 |  →  Hash  (cost=32276.55..32276.55 rows=15689 width=44) (actual ti… |
| 19 |  Buckets: 16384  Batches: 1  Memory Usage: 750kB |
| 20 |  →  HashAggregate  (cost=32080.43..32276.55 rows=15689 width… |
| 21 |  Group Key: document_2.publication_place |
| 22 |  Batches: 1  Memory Usage: 2577kB |
| 23 |  →  Hash Join  (cost=438.00..32001.99 rows=15689 width… |
| 24 |  Hash Cond: (document_2.id_document = picture_1.i… |
| 25 |  →  Seq Scan on document document_2  (cost=0.00.… |
| 26 |  →  Hash  (cost=241.89..241.89 rows=15689 width=… |
| 27 |  Buckets: 16384  Batches: 1  Memory Usage: … |
| 28 |  →  Seq Scan on picture picture_1  (cost=0… |
| 29 | Planning Time: 0.613 ms |




Resultados: FAV esa la mejor opcion por simplicidad y velocidad

## Investigacion sobre otras tecnicas de optimizacion no vistas en clase

Investigacion sobre otras tecnicas de optimizacion no vistas en clase (no puede ser fav, with, index)
1. identificarlas
2. caracterizadas
3. ejemplo

En esta seccion se hablara de algunas tecnicas de optimizaciones adicionales que son muy interesantes y unicas a su manera

- Parallel Queries
- Partitioning
- Materialized Views
- Prepared Statements

Otros son

- Prepared Statements
- indexes

### Parallel Queries

Las consultas paralelas, o "parallel queries", son aquellas consultas SQL que se ejecutan de manera simultánea utilizando múltiples procesadores o núcleos del servidor de base de datos. Esta técnica de optimización se utiliza para acelerar el procesamiento de consultas complejas al dividir la carga de trabajo entre varios hilos de ejecución.

Es el trabajo del planificador de consultas determinar si es la mejor estrategia para usarse y cuantos nucleos puede usar, esto esta definido en la configuracion de postgres bajo la variable

`max_parallel_workers_per_gather`: El numero de workers que el planificador usara

Se caracteriza por:

- **División de Tareas:** Las tareas de la consulta se dividen en subconjuntos independientes que pueden ejecutarse simultáneamente.

- **Uso de Recursos Paralelos:** Se aprovechan múltiples núcleos o procesadores disponibles en el sistema para realizar operaciones en paralelo.
- **Coordinación de Resultados:** Al finalizar las operaciones paralelas, los resultados se combinan para producir el resultado final de la consulta.
- **Eficiencia en Consultas Complejas:** Se beneficia especialmente en consultas que involucran grandes conjuntos de datos o operaciones complejas que pueden dividirse en partes independientes.

### Partitioning

El particionado es una técnica de optimización en bases de datos que consiste en dividir una tabla grande en secciones más pequeñas llamadas particiones. Cada partición tiene su propio conjunto de datos y, al dividir la tabla en partes más manejables, se pueden obtener beneficios en términos de rendimiento y mantenimiento. A continuación, se abordan los puntos clave sobre el particionado en SQL:

PostgreSQL tiene soporte para los siguientes 3 tipos de particiones:

- Range Partitioning
- List Partitioning
- Hash Partitioning

Tiene como caracteristicas:

- **Mejora en el Rendimiento:** Las consultas y operaciones que se centran en un subconjunto específico de datos pueden ser más rápidas, ya que el sistema solo necesita buscar en la partición relevante en lugar de toda la tabla.
- **Facilita el Mantenimiento:** Las operaciones de mantenimiento, como la carga de datos y la eliminación, pueden ser más eficientes al trabajar con particiones más pequeñas en lugar de la tabla completa.
- **Optimización de Recursos:** Al limitar el acceso a ciertas particiones, el sistema puede asignar recursos de manera más efectiva para gestionar operaciones concurrentes.

```
CREATE TABLE mytable (id SERIAL PRIMARY KEY, created_at TIMESTAMP);
CREATE TABLE mytable_2019 PARTITION OF mytable FOR VALUES FROM ('2019-01-01') TO
('2020-01-01');
CREATE TABLE mytable_2020 PARTITION OF mytable FOR VALUES FROM ('2020-01-01') TO
('2021-01-01');
CREATE TABLE mytable_2021 PARTITION OF mytable FOR VALUES FROM ('2021-01-01') TO
('2022-01-01');
```

Materialized Views

Las vistas materializadas (Materialized Views) son objetos de base de datos que almacenan los resultados de una consulta para permitir un acceso más rápido y eficiente a los datos. A diferencia de las vistas regulares, que son consultas almacenadas y recalculadas cada vez que se llaman, las vistas materializadas almacenan físicamente los resultados y se actualizan periódicamente.

Se caracteriza por:

- **Almacenamiento Físico:** A diferencia de las vistas regulares, las vistas materializadas almacenan físicamente los datos en disco.
- **Actualización Programada:** Los datos en una vista materializada se actualizan periódicamente según una programación o en respuesta a eventos específicos.
- **Acceso Rápido:** Proporcionan un acceso más rápido a los resultados precalculados en comparación con la ejecución de la consulta original.

```
CREATE MATERIALIZED VIEW mv_ventas_mensuales AS
SELECT
    vendedor_id,
    EXTRACT(MONTH FROM fecha) AS mes,
    SUM(monto) AS total_ventas
FROM
```

```
    ventas
GROUP BY
    vendedor_id, EXTRACT(MONTH FROM fecha);
```

## Bibliografia


- PostgreSQL Documentacion
  - Parallel Querys
  - Partitioning
  - Materialized Views
- Blog Article
  - PostgreSQL Performance: Top 10 Optimization Tips | Medium
- Youtube Video
  - PostgreSQL Parallel Queries 12/12 - YouTube


## Conclusiones

Tras abordar distintas problemáticas del caso de estudio utilizando SQL básico, FAV y WITH, se ha observado que, en muchos casos, las window functions (FAV) ofrecen una solución más simple y eficiente. Esta técnica proporciona un rendimiento destacado al

cuando se usan funciones de agregacion en comparacion a no usarla. Además, se ha investigado sobre otras técnicas de optimización, como las consultas paralelas y el particionado de tablas, resaltando su impacto positivo en la mejora del rendimiento de las consultas en bases de datos.

La elección de la estrategia óptima depende de la naturaleza específica de la consulta y de la estructura de los datos. Las consultas paralelas resultan beneficiosas en operaciones complejas, mientras que el particionado de tablas es especialmente útil para mejorar la eficiencia en la gestión de grandes conjuntos de datos. En términos de simplicidad y velocidad, FAV se destaca en varias situaciones seguida por WITH que puede ser mas rapida en determinados casos, estas ofrecen una alternativa robusta para optimizar consultas en SQL.