OPTIMIZACIÓN

A) Si hacemos EXPLAIN de la consulta, nos indica el coste de esta en diferentes partes, como lo que cuesta realizar el filtro, o juntar todo.

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Aggregate (cost=5627.93..5627.94 rows=1 width=8)
2	-> Gather (cost=1000.00..5627.92 rows=2 width=4)
3	Workers Planned: 1
4	-> Parallel Seq Scan on orders (cost=0.00..4627.72 rows=1 width=4)
5	Filter: ((totalamount > '100'::numeric) AND (date part('year'::text

Si creamos dos índices, uno sobre la columna del año y el otro sobre la columna del mes (para los que se usara al hacer el filtro), podemos ver como los costes totales se reducen:

Data Output | Explain | Messages | History

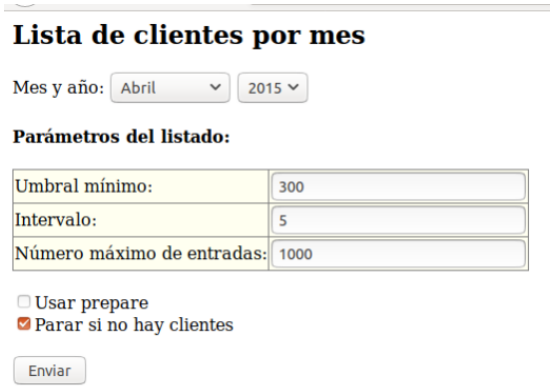
	QUERY PLAN text
1	Aggregate (cost=58.05..58.06 rows=1 width=8)
2	-> Bitmap Heap Scan on orders (cost=38.73..58.05 rows=2 width=4)
3	Recheck Cond: ((date part('month'::text, (orderdate)::timestamp without time z
4	Filter: (totalamount > '100'::numeric)
5	-> BitmapAnd (cost=38.73..38.73 rows=5 width=0)
6	-> Bitmap Index Scan on mes (cost=0.00..19.24 rows=909 width=0)
7	Index Cond: (date part('month'::text, (orderdate)::timestamp with
8	-> Bitmap Index Scan on anno (cost=0.00..19.24 rows=909 width=0)
9	Index Cond: (date part('year'::text, (orderdate)::timestamp witho

Pero creamos un único índice sobre ambas columnas del filtro, y volvemos a hacer un EXPLAIN sobre la consulta, podemos ver como los costes de esta se reducen aún más:

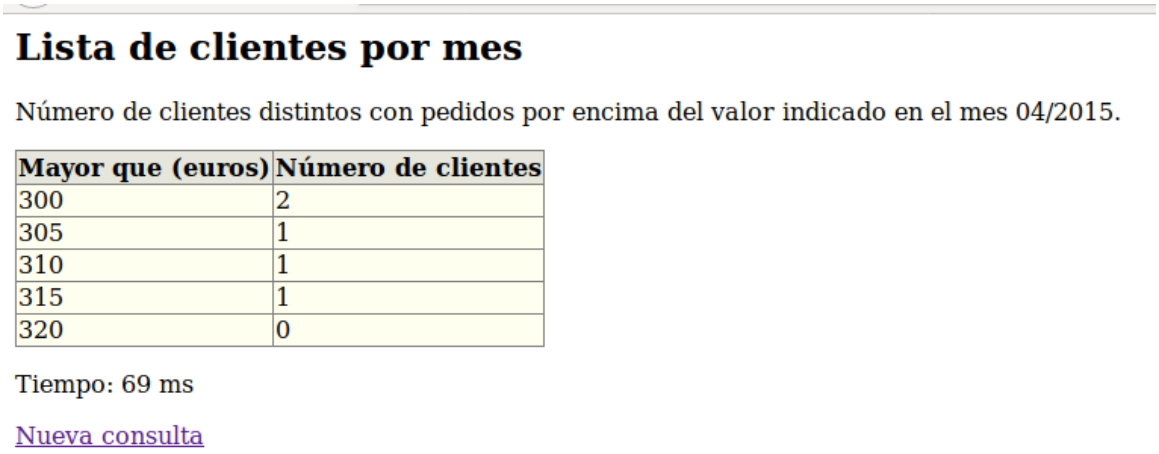
Data Output | Explain | Messages | History

	QUERY PLAN text
1	Aggregate (cost=23.80..23.81 rows=1 width=8)
2	-> Bitmap Heap Scan on orders (cost=4.47..23.79 rows=2 width=4)
3	Recheck Cond: ((date part('year'::text, (orderdate)::timestamp w
4	Filter: (totalamount > '100'::numeric)
5	-> Bitmap Index Scan on anno (cost=0.00..4.47 rows=5 width=0)
6	Index Cond: ((date part('year'::text, (orderdate)::timesta

Por tanto, para esta consulta podemos concluir que lo mejor para reducir costes seria hacer un único índice sobre ambas columnas de la búsqueda.



B) Haciendo la consulta en apache, se ve que el rendimiento se cambia bastante. El tiempo de ejecución mejora con el uso de un índice, pero si el índice se tiene que crear de nuevo al ejecutar el programa, entonces el tiempo de rendimiento es peor.



Lista de clientes por mes

Número de clientes distintos con pedidos por encima del valor indicado en el mes 04/2015.

Mayor que (euros)	Número de clientes
300	2
305	1
310	1
315	1
320	0

Tiempo: 69 ms

Nueva consulta

La imagen de arriba es el tiempo de ejecución sin un índice.

Lista de clientes por mes

Número de clientes distintos con pedidos por encima del valor indicado en el mes 04/2015.

Mayor que (euros)	Número de clientes
300	2
305	1
310	1
315	1
320	0

Tiempo: 307 ms

Nueva consulta

La imagen de arriba es el tiempo de ejecución con la creación de un índice en el PREPARE.

Lista de clientes por mes

Número de clientes distintos con pedidos por encima del valor indicado en el mes 04/2015.

Mayor que (euros)	Número de clientes
300	2
305	1
310	1
315	1
320	0

Tiempo: 17 ms

Nueva consulta

La imagen de arriba es el tiempo de ejecución con el índice ya creado.

Se ve que el mejor rendimiento es el que tiene un índice ya creado. El índice se hace sobre el año y mes. Otros índices se podrían hacer sobre el mes solo o el año solo, pero el mejor índice se hace juntando las dos cosas.

C)

i) Nada más ejecutarse, la primera consulta devuelve los resultados. Se ve en las imagenes de abajo.

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Seq Scan on customers (cost=3961.65..4490.81 rows=7046 width=4)
2	Filter: (NOT (hashed SubPlan 1))
3	SubPlan 1
4	-> Seq Scan on orders (cost=0.00..3959.38 rows=909 width=4)
5	Filter: ((status)::text = 'Paid'::text)

Data Output | Explain | Messages | History

	QUERY PLAN text
1	HashAggregate (cost=4537.41..4539.41 rows=200 width=4)
2	Group Key: customers.customerid
3	Filter: (count(*) = 1)
4	-> Append (cost=0.00..4462.40 rows=15002 width=4)
5	-> Seq Scan on customers (cost=0.00..493.93 rows=14093 width=4)
6	-> Seq Scan on orders (cost=0.00..3959.38 rows=909 width=4)
7	Filter: ((status)::text = 'Paid'::text)

Data Output | Explain | Messages | History

	QUERY PLAN text
1	HashSetOp Except (cost=0.00..4640.83 rows=14093 width=8)
2	-> Append (cost=0.00..4603.32 rows=15002 width=8)
3	-> Subquery Scan on "*SELECT* 1" (cost=0.00..634.86 rows=14093 width=8)
4	-> Seq Scan on customers (cost=0.00..493.93 rows=14093 width=4)
5	-> Subquery Scan on "*SELECT* 2" (cost=0.00..3968.47 rows=909 width=8)
6	-> Seq Scan on orders (cost=0.00..3959.38 rows=909 width=4)
7	Filter: ((status)::text = 'Paid'::text)

ii) La segunda consulta, que usa consultas anidadas con uniones es la que más se beneficia de la ejecución en paralelo.

D) El generador de estadísticas muestra los movimientos de la base de datos, las cosas que se haya hecho, y el tamaño que ocupan ciertos aspectos, como el índice. La planificación de las dos consultas es la misma hasta que se generan las estadísticas porque la cantidad de sequential scans y scans usando índices es diferente dependiendo de la consulta.

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Aggregate (cost=3507.17..3507.18 rows=1 width=8)
2	-> Seq Scan on orders (cost=0.00..3504.90 rows=909 width=0)
3	Filter: (status IS NULL)

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Aggregate (cost=3961.65..3961.66 rows=1 width=8)
2	-> Seq Scan on orders (cost=0.00..3959.38 rows=909 width=0)
3	Filter: ((status)::text = 'Shipped'::text)

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Aggregate (cost=1496.52..1496.53 rows=1 width=8)
2	-> Bitmap Heap Scan on orders (cost=19.46..1494.25 rows=909 width=0)
3	Recheck Cond: (status IS NULL)
4	-> Bitmap Index Scan on estado (cost=0.00..19.24 rows=909 width=0)
5	Index Cond: (status IS NULL)

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Aggregate (cost=1498.79..1498.80 rows=1 width=8)
2	-> Bitmap Heap Scan on orders (cost=19.46..1496.52 rows=909 width=0)
3	Recheck Cond: ((status)::text = 'Shipped'::text)
4	-> Bitmap Index Scan on estado (cost=0.00..19.24 rows=909 width=0)
5	Index Cond: ((status)::text = 'Shipped'::text)

Table statistics report - orders

Generated: vie 07 dic 2018 11:31:54 CET
Server: local (local:.s.PGSQL.5432)
Database: si1
Schema: public

Table statistics

Statistic	Value
Sequential Scans	34
Sequential Tuples Read	5684207
Index Scans	0
Index Tuples Fetched	583666
Tuples Inserted	181790
Tuples Updated	0
Tuples Deleted	0
Tuples HOT Updated	0
Live Tuples	181790
Dead Tuples	0
Heap Blocks Read	1689
Heap Blocks Hit	337163
Index Blocks Read	2
Index Blocks Hit	16
Toast Blocks Read	0
Toast Blocks Hit	0
Toast Index Blocks Read	0
Toast Index Blocks Hit	0
Last Vacuum	
Last Autovacuum	
Last Analyze	2018-12-07 11:30:31.242431+01
Last Autoanalyze	
Vacuum counter	0
Autovacuum counter	0
Analyze counter	3
Autoanalyze counter	0
Table Size	13 MB
Toast Table Size	8192 bytes
Indexes Size	8344 kB

Report generated by pgAdmin III