MEMORIA PRACTICA 3

Apartado E

E.b En la siguiente captura podemos ver el resultado al ejecutar EXPLAIN con la consulta del ejercicio. (Sin aplicar índices)

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 o (cost=0.00..4627.72 rows=1 width=4)
5	Filter: (('100'::numeric <= totalamount) AND (date part('year'::text, (orderdate)::timestamp without time zone) = '2015'::double precision) AND (date part('month'::text, (orderdate)::timestamp without time zon

Como vemos, podemos ver en la primera línea (Aggregate), el coste de la consulta y el número de filas que tiene el resultado. Vemos que en la query se realiza un bucle en el que se recorre paralelamente y secuencialmente la tabla orders, realizando las comparaciones con los requisitos WHERE de la consulta a través de los que se filtra.

E.d Ahora vamos a ver el resultado aplicando el índice sobre el año y el mes del pedido (ord_orders)

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 o (cost=4.47..23.79 rows=2 width=4)
3	Recheck Cond: ((date part('year'::text, (orderdate)::timestamp without time zone) = '2015'::double precision) AND (date part('month'::text, (orderdate)::timestamp without time zone) = '4'::double precision))
4	Filter: ('100'::numeric <= totalamount)
5	-> Bitmap Index Scan on ord orders (cost=0.00..4.47 rows=5 width=0)
6	Index Cond: ((date part('year'::text, (orderdate)::timestamp without time zone) = '2015'::double precision) AND (date part('month'::text, (orderdate)::timestamp without time zone) = '4'::double precision))

Podemos apreciar que el coste de ejecución de la misma consulta ahora con índice es mucho menor, este índice que hemos creado filtra la tabla primero mediante el año y mes que nos interesa (bitmap heap scan) y por lo tanto después en el bucle que realiza las comparaciones para seleccionar los clientes solo realizará las comparaciones con los elementos indexados de arriba (de ahi que se reduzca considerablemente el tiempo de ejecución).

E.e Vamos a probar ahora otros dos nuevos índices que serán con únicamente uno de los extract, el primero para el año y el segundo para el mes (ord_year, ord_month respectivamente)

1.

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Aggregate (cost=1509.94..1509.95 rows=1 width=8)
2	-> Bitmap Heap Scan on orders o (cost=19.24..1509.93 rows=2 width=4)
3	Recheck Cond: (date part('year'::text, (orderdate)::timestamp without time zone) = '2015'::double precision)
4	Filter: (('100'::numeric <= totalamount) AND (date part('month'::text, (orderdate)::timestamp without time zone) = '4'::double precision))
5	-> Bitmap Index Scan on ord year (cost=0.00..19.24 rows=909 width=0)
6	Index Cond: (date part('year'::text, (orderdate)::timestamp without time zone) = '2015'::double precision)

2.

Output pane

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Aggregate (cost=1509.94..1509.95 rows=1 width=8)
2	-> Bitmap Heap Scan on orders o (cost=19.24..1509.93 rows=2 width=4)
3	Recheck Cond: (date part('month'::text, (orderdate)::timestamp without time zone) = '4'::double precision)
4	Filter: (('100'::numeric <= totalamount) AND (date part('year'::text, (orderdate)::timestamp without time zone) = '2015'::double precision))
5	-> Bitmap Index Scan on ord month (cost=0.00..19.24 rows=909 width=0)
6	Index Cond: (date part('month'::text, (orderdate)::timestamp without time zone) = '4'::double precision)

Observamos que estos índices hacen que la consulta tenga un coste similar entre ellos sin embargo superior a cuando actúan los dos en el mismo (aunque sigue siendo bastante inferior a la ejecución de la consulta sin índices), esto es debido a que ahora se realizan más comparaciones en el bucle con pedidos que no nos interesan por estar en anos o meses distintos respectivamente.

Apartado F

El resultado de analizar las tres consultas es el siguiente:

1.

Output pane

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)

2.

Output pane

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)

3.

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)

F.a Vemos que en la primera consulta lo que se realiza es una búsqueda previa guardando en otra estructura los clientes que estuvieran en orders con algún pedido pagado (recorriendo toda la tabla orders) y después se recorre la tabla customers filtrando por aquellos clientes que no estuvieran en la estructura de la subconsulta anterior, vemos que recorremos de manera completa dos tablas por lo que tiene un coste elevado.

En la segunda consulta se realiza una unión entre todos los customerid de la tabla customers y los customerid de la tabla orders que tengan el status a 'Paid', posteriormente se agrupa por customerid eligiendo sólo aquellos que aparecen una única vez en la unión (que son los que no tienen ningún pedido pagado, ya que los que tienen algún pedido con status = 'Paid' aparecen mínimo dos veces), por lo tanto por la cantidad de acciones que realiza esta consulta vemos que es la que tiene el coste más elevado de las tres.

En la tercera se realiza una consulta basada en un except, esta se beneficia del recorrido de dos tablas en paralelo ya que únicamente analiza aquellos customerid que no esten en la segunda subconsulta, por lo tanto es la que menos coste tiene

i. La consulta número 3 , ya que el primer número que aparece en la primera fila después de cost nos dice el coste aproximado del tiempo que se tarda hasta devolver el primer resultado.
ii. Vemos que las consultas 2 y 3 se pueden aprovechar del paralelismo. Esto es debido a que en la segunda se pueden hacer las dos consultas en paralelo (las consultas que unen mediante UNION). Y en la consulta 3 se aprovecha el paralelismo ya que la primera consulta va analizando los customerid que no están en la segunda subconsulta.

Apartado G

G.b Análisis de las consultas sin ningun indice:
1.

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)

2.

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)

Vemos que en ambas consultas se realiza un recorrido de la tabla entera orders filtrando en cada una de ellas según la condición de la consulta, el coste es elevado puesto que se recorre la tabla entera en ambos casos. Podemos apreciar que la planificación de las consultas es la misma si no tenemos índices.

G.d Veamos ahora lo que ocurre cuando añadimos un índice a la columna ‘status’:

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 ord status (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 ord status (cost=0.00..19.24 rows=909 width=0)
5	Index Cond: ((status)::text = 'Shipped'::text)

Lo que observamos es que ahora los costes de las consultas son menos elevados debido a que en ambas el recorrido de la tabla orders de comparaciones se realiza únicamente en aquellos pedidos que tienen el status igual al que estamos buscando en la condición de la query, recorriendo así muchos menos casos en el bucle. La planificación de las dos consultas siguen siendo iguales, ya que el planificador no dispone de las estadísticas de la tabla (no sabe el número de pedidos con status a NULL, ni cuantos Shipped, etc).

G.f y G.g

Probemos ahora a analizar las consultas una vez hemos hecho el analyze de la tabla orders:

1. con NULL:

	QUERY PLAN text
1	Aggregate (cost=7.28..7.29 rows=1 width=8) (actual time=0.017..0.017 rows=1 loops=1)
2	-> Index Only Scan using ord status on orders (cost=0.42..7.27 rows=1 width=0) (actual time=0.015..0.015 rows=0 loops=1)
3	Index Cond: (status IS NULL)
4	Heap Fetches: 0
5	Planning time: 0.063 ms
6	Execution time: 0.044 ms

2. Con ‘Paid’:

	QUERY PLAN text
1	Aggregate (cost=2299.62..2299.63 rows=1 width=8) (actual time=7.049..7.050 rows=1 loops=1)
2	-> Bitmap Heap Scan on orders (cost=348.75..2255.64 rows=17591 width=0) (actual time=1.399..5.851 rows=18163 loops=1)
3	Recheck Cond: ((status)::text = 'Paid'::text)
4	Heap Blocks: exact=1686
5	-> Bitmap Index Scan on ord status (cost=0.00..344.35 rows=17591 width=0) (actual time=1.138..1.138 rows=18163 loops=1)
6	Index Cond: ((status)::text = 'Paid'::text)
7	Planning time: 0.070 ms
8	Execution time: 7.082 ms

3. Con ‘Processed’

	QUERY PLAN text
1	Aggregate (cost=2985.04..2985.05 rows=1 width=8) (actual time=10.605..10.605 rows=1 loops=1)
2	-> Bitmap Heap Scan on orders (cost=737.85..2891.68 rows=37346 width=0) (actual time=2.028..8.543 rows=36304 loops=1)
3	Recheck Cond: ((status)::text = 'Processed'::text)
4	Heap Blocks: exact=1685
5	-> Bitmap Index Scan on ord status (cost=0.00..728.51 rows=37346 width=0) (actual time=1.849..1.849 rows=36304 loops=1)
6	Index Cond: ((status)::text = 'Processed'::text)
7	Planning time: 0.105 ms
8	Execution time: 10.657 ms

4. Con ‘Shipped’:

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Finalize Aggregate (cost=4210.35..4210.36 rows=1 width=8) (actual time=92.351..92.352 rows=1 loops=1)
2	-> Gather (cost=4210.24..4210.35 rows=1 width=8) (actual time=92.189..96.539 rows=2 loops=1)
3	Workers Planned: 1
4	Workers Launched: 1
5	-> Partial Aggregate (cost=3210.24..3210.25 rows=1 width=8) (actual time=84.323..84.324 rows=1 loops=2)
6	-> Parallel Seq Scan on orders (cost=0.00..3023.69 rows=74619 width=0) (actual time=0.476..80.300 rows=63662 loops=2)
7	Filter: ((status)::text = 'Shipped'::text)
8	Rows Removed by Filter: 27234
9	Planning time: 3.414 ms
10	Execution time: 97.285 ms

Observamos que para tanto ‘Paid’ como ‘Processed’ una vez hemos hecho el analyze la consulta se hace utilizando el índice creado en la columna status, esto es así ya que al no

haber una gran cantidad de ellos resulta más rentable utilizar el índice y no recorrer toda la tabla de orders, sin embargo, para Shipped como hay muchos pedidos de este tipo la forma más eficiente es recorrer la tabla de forma paralela puesto que la diferencia entre el número de pedidos total y numero de pedidos con status = ‘Shipped’ no es tan grande como para que resulte rentable indexar la tabla.
Por otro lado tenemos la consulta cuando los pedidos son null que al no haber ninguno en nuestra base de datos realiza el índice simple para no recorrer toda la tabla.
Por lo tanto vemos que esta última es la consulta que tiene menos coste, despues estan practicamente a la par Processed y Paid ya que utilizan índices y por ultimo la que más tarda es Shipped (al tener también un mayor número de elementos y no utilizar índices)