Análisis de consultas

Requerimiento Funcional de Consulta 9

RFC9-Se quiere conocer la información de los clientes que consumieron al menos una vez un determinado servicio del hotel, en un rango de fechas. Los resultados deben ser clasificados según un criterio deseado por quien realiza la consulta. En la clasificación debe ofrecerse la posibilidad de agrupamiento y ordenamiento de las respuestas según los intereses del usuario que consulta como, por ejemplo, por los datos del cliente, por fecha y número de veces que se utilizó el servicio. Esta operación está disponible para el recepcionista del hotel, el gerente del hotel y también para los organizadores de eventos.

Sentencia SQL

```
Select us.nombre, us.identificacion, count(*)
from usuario us
inner join reserva res
on us.identificacion=res.id_usuario
inner join apartan ap
on res.id=ap.idreserva
inner join
habitacion hab
on ap.numerohabitacion=hab.numerohabitacion
inner join sirven sirv
on hab.numerohabitacion=sirv.numerohabitacion
where sirv.FECHAUSO between 2019010101 and 2019070101
and sirv.IDSERVICIO=1
group by us.identificacion, us.nombre;
```

Para solucionar el requerimiento funcional 9, se usó la sentencia SQL mostrada en la imagen 1. Se utilizaron las tablas usuario, reserva, apartan, habitaciones y sirven, los cuales contenían la información necesaria para resolver el requerimiento funcional. La tabla usuario, contiene 9001 registros, la tabla reserva contiene 10000 registros, la tabla apartan contiene 53412 registros, la tabla habitación, contiene 100 registros y la tabla sirven contiene 200000 registros. A raíz de que la tabla sirven contiene una gran cantidad de registros, y dentro de la consulta hay una condición

de búsqueda sobre la fecha de uso, se decidió crear un índice secundario, de un árbol B+, ya que este nos facilita las consultas en rangos.

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304A141910	FECHAUSO	ISIS2304A141910	SIRVEN	FECHAUSO	1	ASC

	Nombre	Valor
1	OWNER	ISIS2304A141910
2	INDEX_NAME	FECHAUSO
3	TABLE_OWNER	ISIS2304A141910
4	TABLE_NAME	SIRVEN
5	PARTITION_NAME	(null)
6	PARTITION_POSITION	(null)
7	SUBPARTITION_NAME	(null)
8	SUBPARTITION_POSITION	(null)
9	OBJECT_TYPE	INDEX
10	BLEVEL	2
11	LEAF_BLOCKS	503
12	DISTINCT_KEYS	8556
13	AVG_LEAF_BLOCKS_PER_KEY	1
14	AVG_DATA_BLOCKS_PER_KEY	22
15	CLUSTERING_FACTOR	195400
16	NUM_ROWS	200000
17	AVG_CACHED_BLOCKS	(null)
18	AVG_CACHE_HIT_RATIO	(null)
19	SAMPLE_SIZE	200000
20	LAST_ANALYZED	18/05/19
21	GLOBAL_STATS	YES
22	USER_STATS	NO
23	STATTYPE_LOCKED	(null)
24	STALE_STATS	NO
25	SCOPE	SHARED

Adicionalmente se realizaron los cálculos del índice para determinar si es viable y cabe en memoria: Se sabe que hay 503 bloques * 2400 que es lo que pesa un bloque, lo que es igual a 1207200 bytes lo que es muy inferior a la RAM.

Adicionalmente en la consulta se usó el índice de la llave primaria de servicio, ya que se debía hacer un index scan sobre el mismo para encontrar solo los que contienen el id del servicio solicitado.

Por otra parte, los valores usados en la consulta, son la fecha inicial del uso, que equivale a 2019010101 y la fecha final de uso que equivale a 2019070101 de un servicio con identificación 1.

Según los parámetros ingresados, el tiempo de ejecución de la sentencia es el siguiente:

```
Se han recuperado 50 filas en 0,034 segundos
```

Al cambiar los parámetros de consulta en diferentes ocasiones, aumentando el rango de búsqueda de las fechas y disminuyéndolo, se observó que el tiempo de ejecución de la sentencia no variaba y se mantenía constante, oscilaba entre los 0,02 y 0,06 segundos. Los datos de prueba son los siguientes.

2019010101	2019050101	3	0,034
2018050101	2019020101	5	0,055
2019030101	2019040101	9	0,035

El plan de consulta ofrecido por Oracle es el siguiente

Primero hace un hash join entre las tablas reserva y usuario, debido a que ambas tablas no caben en memoria, y se busca encontrar las llaves que coincidan. Después, se hace un nested loop join, usando los índices de la tabla apartan, los cuales fueron encontrados mediante un index scan. La tabla resultante del nested loop join se le hace un hash join con la tabla o la vista resultante de la selección de las tuplas con el parámetro de fecha determinado.

Plan de consulta desarrollado

```
Select Statement
        Hash Join
                Usuario                                 Full Table Scan

                Hash Join
                        Reserva                         Full Table Scan

                        Nested Loop Join
                                Apartan                 Full Table Scan

                                Hash Join
                                        Sirven          Index Scan
                                                Filter Predicate on id=1, 2019010101<=FechaUso<=2019050101
                                        Habitacion      Full Table Scan
```

El plan de consulta desarrollado por el grupo de trabajo es muy similar al planteado por Oracle. En un principio, se hace un index scan a la tabla sirven, ya que se debe encontrar las tuplas con ServicioId igual a 1. Por otro lado, toca leer todas las tuplas de habitación, para poder el join. Se realiza un Hash join, debido, a que toca juntarlos por una llave igual, esto genera que las colisiones sean correctas. Seguido se hace un Nested Loop Join entre la tabla resultante del hash anterior y la tabla apartan, la cual tiene que leerse completa en lectura secuencial rápida, cabe resaltar que al realizar el Nested Loop Join, significa que una de las tablas cabe en memoria. Después se hacen dos hashes joins, uno con la tabla reservas y otro con la tabla usuarios.

Requerimiento Funcional de Consulta 10

RFC-10Se quiere conocer la información de los clientes que NO consumieron ninguna vez un determinado servicio del hotel, en un rango de fechas. Los resultados deben ser clasificados según un criterio deseado por quien realiza la consulta. En la clasificación debe ofrecerse la posibilidad de agrupamiento y ordenamiento de las respuestas según los intereses del usuario que consulta como, por ejemplo, por los datos del cliente, por fecha y número de veces que se utilizó el servicio. Esta

operación está disponible para el recepcionista del hotel, el gerente del hotel y también para los organizadores de eventos.

Sentencia SQL

```
select *
from usuario
where identificacion not in(
Select us.identificacion
from usuario us
inner join reserva res
on us.identificacion=res.id_usuario
inner join apartan ap
on res.id=ap.idreserva
inner join
habitacion hab
on ap.numerohabitacion=hab.numerohabitacion
inner join sirven sirv
on hab.numerohabitacion=sirv.numerohabitacion
where sirv.FECHAUSO between 2019010101 and 20190501
and sirv.IDSERVICIO=1);
```

Para solucionar el requerimiento funcional de consulta 10, se tuvo en cuenta el requerimiento anterior, debido a que este es una modificación, donde se buscaba encontrar los usuarios que no consumieron un servicio determinado en el hotel. Por lo que primero se decidió encontrar los usuarios que, si consumieron, y seguido, comparar las identificaciones de los usuarios, los cuales no se encuentran dentro de los que si consumieron un servicio. Se tiene la misma cantidad anteriormente mencionada de registros en las tablas.

Este requerimiento funcional uso el índice anteriormente creado por el requerimiento funcional uno, para encontrar los rangos de fechas dados. Se sabe que el índice ocupa 503 bloques por lo que si son óptimos para su uso debido a que caben en memoria principal. Por lo que es un índice secundario disperso, usara un árbol b+ como estructura de datos, con apuntadores a memoria.

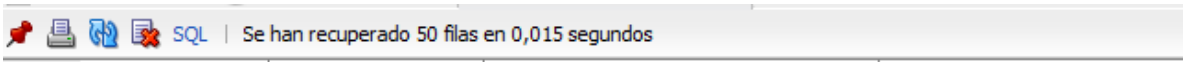
Como este requerimiento funcional es una extensión del anterior, se usó el mismo índice primario en la tabla apartan, donde se busca en un rango los servicios con un id pasado por parámetro.

Para verificar que los datos si concuerdan y el requerimiento funciona correctamente, se van a usar los mismos valores de prueba del requerimiento

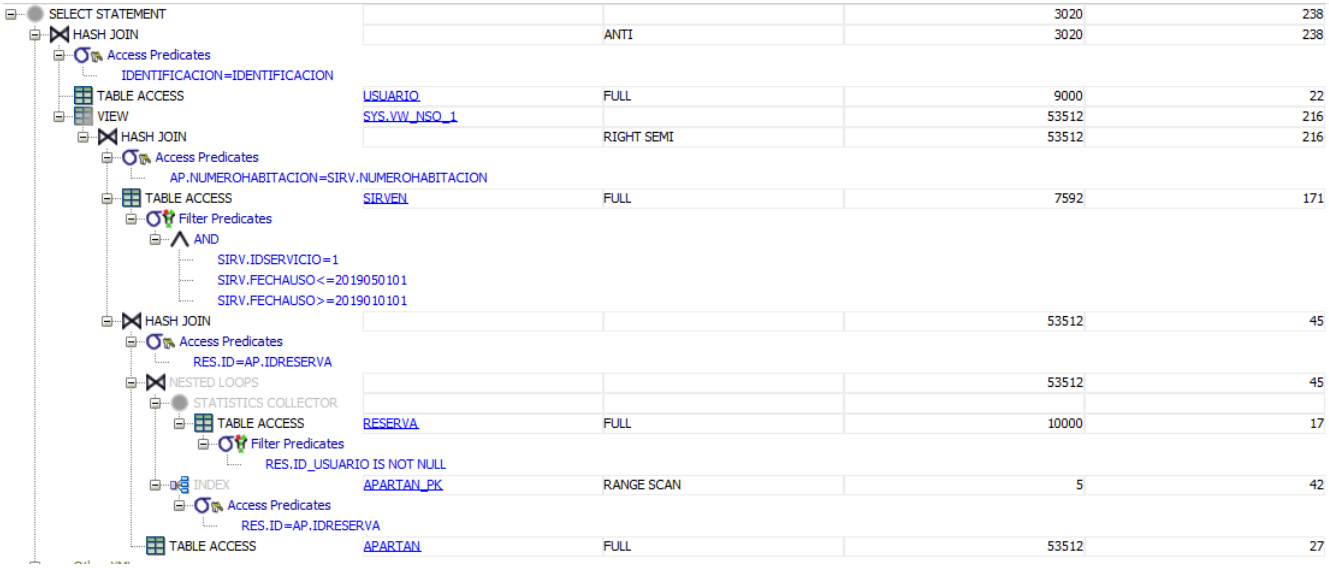
anterior, esto para observar que dichos valores no concuerdan, y que los valores presentados en el requerimiento anterior, no están presentes en este requerimiento.

2019010101	2019050101	3	0,032
2018050101	2019020101	5	0,04
2019030101	2019040101	9	0,022

Después de analizar el tiempo de consulta, se puede evidenciar que es más eficiente esta búsqueda que el requerimiento funcional anterior, sin importar si se ejecuta una operación extra para verificar los usuarios que no pertenecen al listado anterior. Su tiempo oscila entre 0,04 segundos y 0,01 segundos.



A continuación, se explica el plan de ejecución, y el porqué de que esta consulta es más eficiente que la anterior.



En el plan de consulta anterior se puede evidenciar que la opción del join varia a la del requerimiento funcional anterior. En un principio, el hash join entre la tabla sirven y el hash join de las tablas reserva y apartan, es de tipo right semi, lo cual significa que retorna la outter table, en la cual es la resultante de todos los registros que hicieron pareja en un inner join, al menos una vez. Esto sirve para que en el último hash join, el cual es de tipo anti, pueda obtener todas las identificaciones que no están dentro del right semi hash join. Por lo que retornara los registros que no entraron en el primer inner join. Como se puede observar, el select statement, va a retornar 3020 tuplas mientras que en requerimiento anterior va a retornar 5981

tuplas, por lo que su tiempo va a ser ligeramente mayor. Adicionalmente la suma de ambos resultados va a dar el total de tuplas de usuarios, por lo que se puede confirmar el éxito de la operación.

Plan de consulta propuesto

```
Select Statement
        Hash Join
                Usuario                                 Full Table Scan

                Hash Join
                        Reserva                         Full Table Scan

                        Nested Loop Join
                                Apartan                 Full Table Scan

                                Hash Join
                                        Sirven          Index Scan
                                                Filter Predicate on id=1, 2019010101<=FechaUso<=2019050101
                                        Habitacion      Unique Scan
```

El plan de consulta desarrollado por el grupo de trabajo es muy similar al planteado por Oracle. En un principio, se hace un index scan a la tabla sirven, ya que se debe encontrar las tuplas con ServicioId igual a 1. Por otro lado, toca leer todas las tuplas de habitación, para poder el join. Se realiza un Hash join, debido, a que toca juntarlos por una llave igual, esto genera que las colisiones sean correctas. Seguido se hace un Nested Loop Join entre la tabla resultante del hash anterior y la tabla apartan, la cual tiene que leerse completa en lectura secuencial rápida, cabe resaltar que al realizar el Nested Loop Join, significa que una de las tablas cabe en memoria. Después se hacen dos hashes joins, uno con la tabla reservas y otro con la tabla usuarios. A diferencia de la consulta anterior, se realiza un anti join, para determinar los usuarios que no pertenecen a dicha consulta, es decir al universo de usuarios, se le restan los usuarios pertenecientes a la consulta.

Requerimiento Funcional de Consulta 11

RFC11-Muestra para cada semana del año (sábado a sábado) el servicio más consumido, el servicio menos consumido, las habitaciones más solicitadas y las habitaciones menos solicitadas. Esta operación es realizada por el gerente general de HotelAndes.

Este requerimiento se divide en dos partes. El primero, encontrar los servicios más y menos consumidos tanto por huéspedes como por participantes de una convención. El segundo, encontrar las habitaciones más y menos solicitadas.

Para encontrar los servicios, se utilizan entonces las tablas de Servicios, Sirven, Servicios_Convencion y Reserva_Convenciones.

El número de entradas de cada tabla es el siguiente:

Servicios: 10

Sirven: 200.000

Servicios_Convencion: 5.000

Reserva_Convenciones: 5.000

Dado que la tabla Sirven maneja un número elevado de registros, podría plantearse un escenario donde dicha tabla tuviera un índice. Afortunadamente este índice fue creado en un punto anterior, por lo que ya se encuentra disponible para su uso. Es un índice secundario de forma B+. Nos resulta realmente práctico ya que esta consulta es por rango de fechas, y el índice es sobre el atributo fechaUso.

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304A141910	FECHAUSO	ISIS2304A141910	SIRVEN	FECHAUSO	1	ASC

Imagen. Muestra al índice que respalda esta consulta.

Debido a que las demás tablas son relativamente pequeñas, se decidió no crear índices sobre ellas. Se utilizarón además los índices de llave primaria de las tablas.

Dado que esta es una consulta por rango, y en el caso específico del requerimiento es un rango semanal, los parámetros de entrada son fechas de la forma AAAAMMDD. La consulta retorna entonces una tabla con todos los servicios usados en ese rango, más el número de veces que fueron utilizados. Ya que la tabla resultante es pequeña y cabe en memoria, se realiza allá la selección del maximo y del mínimo.

```
SELECT TOT.ID, TOT.NOMBRE, SUM(NUMVECES) AS TOTALVECES
FROM (
        SELECT RESP.ID, RESP.NOMBRE, SUM(RESP.NUMEROVECES) AS NUMVECES
        FROM(
                SELECT SER.ID, COUNT(SER.ID) AS NUMEROVECES, SER.NOMBRE,  SIR.FECHAUSO
                FROM SERVICIOS SER, SIRVEN SIR
                WHERE SER.ID = SIR.IDSERVICIO AND SIR.FECHAUSO BETWEEN '2019051200' AND '2019051823'
                GROUP BY SIR.FECHAUSO, SER.NOMBRE, SER.ID
            )RESP
        GROUP BY RESP.ID, RESP.NOMBRE

        UNION ALL

        SELECT SER.ID, SER.NOMBRE, COUNT(RESC.ID) AS NUMEROVECES
        FROM SERVICIOS SER, SERVICIOS_CONVENCION SERC, RESERVA_CONVENCIONES RESC
        WHERE SER.ID = SERC.ID_RESERVA AND SERC.ID_RESERVA = RESC.ID
         AND ((RESC.FECHAINICIO BETWEEN '20190512' AND '20190518') OR (RESC.FECHAFIN BETWEEN '20190512' AND '20190518'))
        GROUP BY SER.ID, SER.NOMBRE
    )TOT
GROUP BY TOT.ID, TOT.NOMBRE
ORDER BY TOTALVECES DESC
```

Imagen. Muestra la sentencia sql que respalda la consulta.

Según los parámetros ingresados, el resultado de ejecutar esta consulta es el siguiente:

SQL | All Rows Fetched: 9 in 0,046 seconds

	ID	NOMBRE	TOTALVECES
1	5	Supermercadosurtimax	446
2	8	Sala reuniones 1	436
3	7	Spa tailandia	431
4	1	Piscina 1	423
5	3	Bar latino	412
6	2	Gimnasio 1	403
7	4	Restaurante italiano	396
8	6	Arturo Calle	394
9	9	Sala conferencias 1	391

Imagen. Muestra los resultados de esta consulta (1 semana).

Para las habitaciones se sigue un proceso bastante similar. En este caso, se usan las tablas Apartan, Reserva, Habitacion_Convencion y Reserva_Convenciones.

El número de entradas de cada tabla es el siguiente:

Apartan: 53.512

Reserva: 10.000

Habitacion_Convencion: 50.000

Reserva_Convenciones: 5000

Se observa que para este caso, encontramos dos tablas con un tamaño importante. Sin embargo, estas tablas ya contienen índices de llave primaria. Adicional a los índices primarios, y dado que se generaron índices secundarios en puntos anteriores, se escoge utilizar los índices por identificador de Apartan, por Fecha de reserva (para apoyar esta búsqueda por rango).

Del mismo modo, como la tabla resultante es pequeña y cabe en memoria, la selección del máximo y el mínimo se realiza en lógica.

```sql
SELECT RESP.NUMEROHABITACION, COUNT(ID) AS NUMEROVECES
FROM(
        SELECT AP.NUMEROHABITACION, RES.ID, RES.FECHAINICIO, RES.FECHAFIN
        FROM APARTAN AP, RESERVA RES
        WHERE AP.IDRESERVA = RES.ID
            AND ((RES.FECHAINICIO BETWEEN '20190512' AND '20190518') OR (RES.FECHAFIN BETWEEN '20190512' AND '20190518'))

        UNION ALL

        SELECT HABC.NUMEROHABITACION, RESC.ID, RESC.FECHAINICIO, RESC.FECHAFIN
        FROM HABITACION_CONVENCION HABC, RESERVA_CONVENCIONES RESC
        WHERE HABC.ID_RESERVA = RESC.ID
            AND ((RESC.FECHAINICIO BETWEEN '20190512' AND '20190518') OR (RESC.FECHAFIN BETWEEN '20190512' AND '20190518'))
    ) RESP
GROUP BY RESP.NUMEROHABITACION
ORDER BY NUMEROVECES DESC
```

Imagen. Muestra la sentencia sql que respalda esta consulta.

Según los parámetros ingresados, el resultado de ejecutar esta consulta es el siguiente:

SQL | Fetching next 50 rows in 0,091 seconds

	NUMEROHABITACION	NUMEROVECES
1	4	44
2	86	41
3	52	40
4	36	38
5	17	37
6	40	37
7	34	36
8	49	36
9	21	35
10	33	35
11	76	34
12	30	34
13	93	34
14	64	33
15	67	33

	NUMEROHABITACION	NUMEROVECES
85	55	25
86	54	23
87	72	23
88	61	23
89	78	22
90	3	22
91	39	22
92	81	21
93	2	20
94	71	19
95	10	19
96	79	19
97	19	18
98	23	18
99	80	18
100	9	18

Imagen. Muestra los resultados para esta consulta (1 semana).

Para consultas con un mayor rango (por ejemplo, colocando un rango de un año), se observa que efectivamente se puede llegar a demorar un poco más tiempo en responder, pero es una diferencia practicamente inperceptible.

SQL | All Rows Fetched: 9 in 0,107 seconds

	ID	NOMBRE	TOTALVECES
1	6	Arturo Calle	8679
2	1	Piscina 1	8575
3	8	Sala reuniones 1	8550
4	7	Spa tailandia	8460
5	3	Bar latino	8441
6	9	Sala conferencias 1	8409
7	5	Supermercadosurtimax	8390
8	2	Gimnasio 1	8365
9	4	Restaurante italiano	8320

Imagen. Muestra los resultados para una consulta con rango de 1 año.

SQL | All Rows Fetched: 100 in 0,095 seconds

	NUMEROHABITACION	NUMEROVECES
1	65	844
2	40	826
3	1	820
4	24	820
5	22	820
6	74	813
7	44	813
8	49	810
9	18	810
10	30	809

Imagen. Muestra los resultados para una consulta con rango de 1 año.

Los planes de consulta ofrecidos por Oracle para cada una de las etapas son los siguientes:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFE
SORT		ORDER BY	2	192	
HASH		GROUP BY	2	192	
VIEW			2	190	
UNION-ALL					
HASH		GROUP BY	1	177	
VIEW			1341	175	
HASH		GROUP BY	1341	175	
HASH JOIN			3386	174	
SER.ID=SIR.IDSERVICIO					
TABLE ACCI	SERVICIOS	FULL	10	3	
TABLE ACCI	SIRVEN	FULL	3386	171	
SIR.FECHAUSO<=2019051823 AND SIR.FECHAUSO>=2019051200					
HASH		GROUP BY	1	13	
NESTED LOOPS			1	12	
NESTED LOOPS			4	12	
HASH JOIN			4	8	
Access Predicates					
SER.ID=SERC.ID_RESERVA					
TABLE ACCI	SERVICIOS	FULL	10	3	
TABLE ACCI	SERVICIOS_CONVENCION	FULL	5000	5	
INDEX	PK_RESERVA_CONV	UNIQUE SCAN	1	0	
SERC.ID_RESERVA=RESC.ID					
TABLE ACCESS	RESERVA_CONVENCIONES	BY INDEX ROWID	1	1	
Filter Predicates					
(RESC.FECHAFIN<=20190518 AND RESC.FECHAFIN>=20190512) OR (RESC.FECHAINICIO>=20190512 AND RESC.FECHAINICIO<=20190518)					
Other XML					

Imagen. Plan de ejecución ofrecido por Oracle para los servicios.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFE
SELECT STATEMENT				81	
SORT		ORDER BY	100	81	
HASH		GROUP BY	100	81	
VIEW			2815	79	
UNION-ALL					
HASH JOIN			1563	44	
AP.IDRESERVA=RES.ID					
NESTED LOOPS			1563	44	
STATISTICS COLL					
TABLE ACCESS	RESERVA	FULL	291	17	
Filter Predicates					
(RES.FECHAFIN>=20190512 AND RES.FECHAFIN<=20190518) OR (RES.FECHAINICIO>=20190512 AND RES.FECHAINICIO<=20190518)					
INDEX	APARTAN_PK	RANGE SCAN	5	42	
Access Predicates					
AP.IDRESERVA=RES.ID					
TABLE ACCESS	APARTAN	FULL	53512	27	
HASH JOIN			1252	34	
Access Predicates					
HABC.ID_RESERVA=RESC.ID					
TABLE ACCESS	RESERVA_CONVENCIONES	FULL	125	9	
Filter Predicates					
(RESC.FECHAFIN<=20190518 AND RESC.FECHAFIN>=20190512) OR (RESC.FECHAINICIO>=20190512 AND RESC.FECHAINICIO<=20190518)					
TABLE ACCESS	HABITACION_CONVENCION	FULL	50000	25	

Imagen. Plan de ejecución ofrecido por Oracle para las habitaciones.

Se observa que Oracle selecciona primero las operaciones que puedan tener índices asociados (como las búsquedas en rango, o los índices de llave primaria). Después de esas

selecciones, realiza Joins tanto Nested como de Hash. Al final, realiza el Union-all de las tablas intermedias, realiza el sort y entrega el resultado.

El plan de consulta sugerido por el grupo fue prácticamente el mismo que sugirió Oracle. Apoyados en los índices secundarios de fecha, se realiza la proyección sobre los datos para hacer el filtrado por la fecha de búsqueda, en el rango dado. Después de esto, y dependiendo si las tablas siguen siendo muy grandes como para caber en memoria, se realizan nested loops o hash joins dependiendo del tamaño. Para la implementación, se decidió realizar las consultas por un rango de 1 semana. La aplicación encuentra la fecha actual, y busca el sábado anterior para generar las consultas. A partir del primer sábado que encuentra, realiza 52 consultas (de las 52 semanas de un año calendario) y retorna al usuario la respuesta detallada de dichas consultas. Una muestra del resultado obtenido se muestra a continuación:

Imagen. Resultado obtenido tras la ejecución de la aplicación.

Requerimiento Funcional de Consulta 12

RFC12-Los buenos clientes son de tres tipos: aquellos que realizan estancias (las estancias están delimitadas por un check in y su respectivo check out) en HotelAndes al menos una vez por trimestre, aquellos que siempre consumen por lo menos un servicio costoso (Entiéndase como costoso, por ejemplo, con un precio mayor a $300.000.oo) y aquellos que en cada estancia consumen servicios de SPA o de salones de reuniones con duración mayor a 4 horas. Esta consulta retorna toda la información de dichos clientes, incluyendo aquella que justifica su calificación como buenos clientes. Esta operación es realizada únicamente por el gerente general de HotelAndes

Sentencia SQL

```sql
SELECT us.nombre,us.identificacion,serv.nombre
FROM USUARIO US
INNER JOIN RESERVA RES
ON US.IDENTIFICACION=RES.ID_USUARIO
INNER JOIN APARTAN AP
ON RES.ID=AP.IDRESERVA
INNER JOIN HABITACION HAB
ON AP.NUMEROHABITACION=HAB.NUMEROHABITACION
INNER JOIN SIRVEN SIRV
ON HAB.NUMEROHABITACION=SIRV.NUMEROHABITACION
INNER JOIN SERVICIOS SERV
ON SIRV.IDSERVICIO=SERV.ID
WHERE SERV.COSTO>=20
OR SERV.NOMBRE LIKE'%SPA%'
OR SERV.NOMBRE LIKE '%SALA REUNIONES%'
OR SERV.NOMBRE LIKE '%SALA CONFERENCIAS%'
OR res.fechainicio between 20180401 and 20180701
and res.fechainicio between 20180701 and 20180901
and res.fechainicio between 20180901 and 20190101
group by us.nombre,us.identificacion,serv.nombre
```

Para solucionar el requerimiento de consulta 12 se uso la anterior sentencia sql, la cual usaba las tablas usuario, reserva, apartan, habitación, sirven, servicios, juntándolas mediante joins para poder obtener el resultado deseado, en la siguiente sección, se van a mostrar los tamaños de las tablas usadas para solucionar el requerimiento funcional.

Este requerimiento funcional hace uso de dos índices primarios, el de reservas y el de usuarios para obtener los mismos. Debido a que dichos índices caben en memoria, su costo es de 0 por lo que volvió la consulta lo mas eficiente posible, adicionalmente gracias a esto, se puede usar el nested loop join, el cual es el mas eficiente teniendo una complejidad de la suma de la lectura secuencial rápida de las tablas usadas. A continuación, se muestra la información de ambos índices usados:

Usuario PK Índice

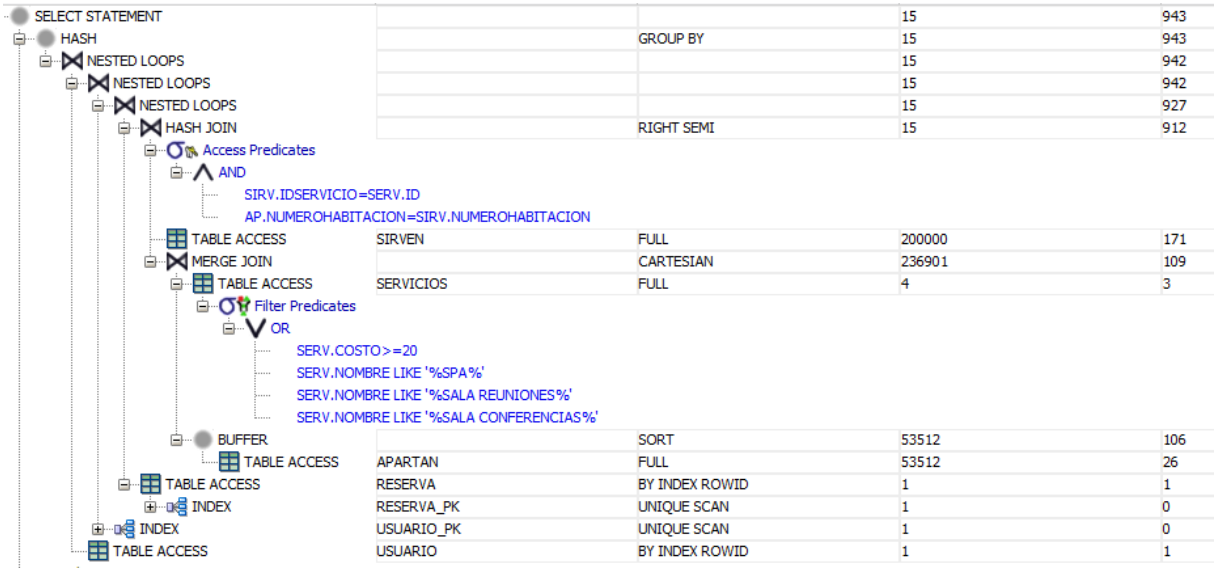
	Nombre	Valor
1	OWNER	ISIS2304A141910
2	INDEX_NAME	USUARIO_PK
3	TABLE_OWNER	ISIS2304A141910
4	TABLE_NAME	USUARIO
5	PARTITION_NAME	(null)
6	PARTITION_POSITION	(null)
7	SUBPARTITION_NAME	(null)
8	SUBPARTITION_POSITION	(null)
9	OBJECT_TYPE	INDEX
10	BLEVEL	1
11	LEAF_BLOCKS	17
12	DISTINCT_KEYS	9000
13	AVG_LEAF_BLOCKS_PER_KEY	1
14	AVG_DATA_BLOCKS_PER_KEY	1
15	CLUSTERING_FACTOR	68
16	NUM_ROWS	9000
17	AVG_CACHED_BLOCKS	(null)
18	AVG_CACHE_HIT_RATIO	(null)
19	SAMPLE_SIZE	9000
20	LAST_ANALYZED	18/05/19
21	GLOBAL_STATS	YES
22	USER_STATS	NO
23	STATTYPE_LOCKED	(null)
24	STALE_STATS	NO
25	SCOPE	SHARED

Reserva PK Índice

	Nombre	Valor
1	OWNER	ISIS2304A141910
2	INDEX_NAME	RESERVA_PK
3	TABLE_OWNER	ISIS2304A141910
4	TABLE_NAME	RESERVA
5	PARTITION_NAME	(null)
6	PARTITION_POSITION	(null)
7	SUBPARTITION_NAME	(null)
8	SUBPARTITION_POSITION	(null)
9	OBJECT_TYPE	INDEX
10	BLEVEL	1
11	LEAF_BLOCKS	18
12	DISTINCT_KEYS	10000
13	AVG_LEAF_BLOCKS_PER_KEY	1
14	AVG_DATA_BLOCKS_PER_KEY	1
15	CLUSTERING_FACTOR	60
16	NUM_ROWS	10000
17	AVG_CACHED_BLOCKS	(null)
18	AVG_CACHE_HIT_RATIO	(null)
19	SAMPLE_SIZE	10000
20	LAST_ANALYZED	13/05/19
21	GLOBAL_STATS	YES
22	USER_STATS	NO
23	STATTYPE_LOCKED	(null)
24	STALE_STATS	NO
25	SCOPE	SHARED

Al ser un índice primario y denso, la estructura de datos que se debería usar es una tabla de hash, para un rápido acceso al registro, mediante un UNIQUE SCAN, de esta manera se accede directamente al registro desde la llave. Esto sucede debido a que las tablas de hash en índices primarios contienen como llave el valor de la columna la cual se decidió el Primary key, y como hoja esta el registro completo. Cabe resaltar que el costo de obtener un registro mediante un índice primario es 0, ya que este esta ubicado directamente en la hoja, en memoria principal.

Plan de ejecución:



En el plan de ejecución mostrado anteriormente se pueden evidenciar varios aspectos. En un principio, se filtran las condiciones de búsqueda sobre servicio, debido a que esta tabla solamente contiene 9 tuplas, por lo que su selección es rápida y no es costosa. Después de filtrar, se hace un merge join entre la tabla apartan y la tabla sirven. Después de la tabla resultante, se hace un hash join con la tabla sirven, debido a que es un join por un foreign key y las tablas no caben en memoria. Seguido se hace un nested loop jooin con la tabla reserva, como la tabla reserva cabe en memoria principal, se usan sus índices para hacer un nested loop join, el cual es el mas eficiente si alguna tabla cabe en memoria. El mismo proceso se lleva acabo con el join entre la tabla resultante anterior y la tabla usuario, debido a que los usuarios caben en memoria, se usan sus índices primarios, para realizar un nested loop join.

Plan de ejecucion

```
Select Statement
        Hash Join
                Usuario                                 Full Table Scan

                Nested Loop
                        Reserva                         Full Table Scan

                        Nested Loop Join
                                Apartan                 Full Table Scan

                                Nested loop
                                        Habitacion      Full table Scan

                                        Merge Join
                                                Sirven  Full Table Scan

                                                Servicio  Unique Scan
                                                        Filter predicate on costo>=20, nombre Like "Sala Conferencias"
```

En el plan de ejecucion planteado por el grupo de trabajo, se puede evidenciar un merge join, esto sucede por que al ser la selectividad muy baja de la tabla servicios, sobre los atributos en la condición del where, resulta mas efectivo que los demás joins. Por otro lado, se pueden evidenciar los nested loops predominando, esto debido a que almenos una de las tablas cabe en memoria, ya que según los bloques que usan, su tamaño es inferior al de la memoria principal. A continuación, se muestran los datos de cada una de las tablas.

Tablas

Apartan

	Nombre	Valor
1	NUM_ROWS	53512
2	BLOCKS	95
3	AVG_ROW_LEN	7
4	SAMPLE_SIZE	53512
5	LAST_ANALYZED	15/05/19
6	LAST_ANALYZED_SINCE	15/05/19

Consumen

	Nombre	Valor
1	NUM_ROWS	81057
2	BLOCKS	244
3	AVG_ROW_LEN	7
4	SAMPLE_SIZE	81057
5	LAST_ANALYZED	18/05/19
6	LAST_ANALYZED_SINCE	18/05/19

Convencion_usuario

Convencion

	Nombre	Valor
1	NUM_ROWS	12745
2	BLOCKS	28
3	AVG_ROW_LEN	8
4	SAMPLE_SIZE	12745
5	LAST_ANALYZED	17/05/19
6	LAST_ANALYZED_SINCE	17/05/19

Habitacion

	Nombre	Valor
1	NUM_ROWS	300
2	BLOCKS	5
3	AVG_ROW_LEN	107
4	SAMPLE_SIZE	300
5	LAST_ANALYZED	13/05/19
6	LAST_ANALYZED_SINCE	13/05/19

Habitacion_convencion

	Nombre	Valor
1	NUM_ROWS	100
2	BLOCKS	5
3	AVG_ROW_LEN	131
4	SAMPLE_SIZE	100
5	LAST_ANALYZED	13/05/19
6	LAST_ANALYZED_SINCE	13/05/19

Habitacion_Mantenimientos

	Nombre	Valor
1	NUM_ROWS	50000
2	BLOCKS	88
3	AVG_ROW_LEN	7
4	SAMPLE_SIZE	50000
5	LAST_ANALYZED	17/05/19
6	LAST_ANALYZED_SINCE	17/05/19

Mantenimientos

	Nombre	Valor
1	NUM_ROWS	100000
2	BLOCKS	370
3	AVG_ROW_LEN	16
4	SAMPLE_SIZE	100000
5	LAST_ANALYZED	17/05/19
6	LAST_ANALYZED_SINCE	17/05/19

Ofrecen

	Nombre	Valor
1	NUM_ROWS	500
2	BLOCKS	13
3	AVG_ROW_LEN	125
4	SAMPLE_SIZE	500
5	LAST_ANALYZED	17/05/19
6	LAST_ANALYZED_SINCE	17/05/19

Planes

	Nombre	Valor
1	NUM_ROWS	30000
2	BLOCKS	50
3	AVG_ROW_LEN	7
4	SAMPLE_SIZE	30000
5	LAST_ANALYZED	17/05/19
6	LAST_ANALYZED_SINCE	17/05/19

Productos

	Nombre	Valor
1	NUM_ROWS	4
2	BLOCKS	5
3	AVG_ROW_LEN	111
4	SAMPLE_SIZE	4
5	LAST_ANALYZED	13/05/19
6	LAST_ANALYZED_SINCE	13/05/19

Reserva

	Nombre	Valor
1	NUM_ROWS	1000
2	BLOCKS	13
3	AVG_ROW_LEN	32
4	SAMPLE_SIZE	1000
5	LAST_ANALYZED	13/05/19
6	LAST_ANALYZED_SINCE	13/05/19

	Nombre	Valor
1	NUM_ROWS	10000
2	BLOCKS	58
3	AVG_ROW_LEN	34
4	SAMPLE_SIZE	10000
5	LAST_ANALYZED	13/05/19
6	LAST_ANALYZED_SINCE	13/05/19

Reserva_Convencion

	Nombre	Valor
1	NUM_ROWS	5000
2	BLOCKS	28
3	AVG_ROW_LEN	27
4	SAMPLE_SIZE	5000
5	LAST_ANALYZED	19/05/19
6	LAST_ANALYZED_SINCE	19/05/19

Servicio_mantenimiento

	Nombre	Valor
1	NUM_ROWS	100000
2	BLOCKS	244
3	AVG_ROW_LEN	7
4	SAMPLE_SIZE	100000
5	LAST_ANALYZED	17/05/19
6	LAST_ANALYZED_SINCE	17/05/19

Servicios

	Nombre	Valor
1	NUM_ROWS	10
2	BLOCKS	5
3	AVG_ROW_LEN	63
4	SAMPLE_SIZE	10
5	LAST_ANALYZED	18/05/19
6	LAST_ANALYZED_SINCE	18/05/19

Servicio_convencion

	Nombre	Valor
1	NUM_ROWS	5000
2	BLOCKS	13
3	AVG_ROW_LEN	7
4	SAMPLE_SIZE	5000
5	LAST_ANALYZED	17/05/19
6	LAST_ANALYZED_SINCE	17/05/19

Sirven

	Nombre	Valor
1	NUM_ROWS	200000
2	BLOCKS	622
3	AVG_ROW_LEN	25
4	SAMPLE_SIZE	200000
5	LAST_ANALYZED	19/05/19
6	LAST_ANALYZED_SINCE	19/05/19

Usuarios

	Nombre	Valor
1	NUM_ROWS	9000
2	BLOCKS	73
3	AVG_ROW_LEN	73
4	SAMPLE_SIZE	9000
5	LAST_ANALYZED	19/05/19
6	LAST_ANALYZED_SINCE	19/05/19