Documento de informe Iteración 2

Sistemas transaccionales

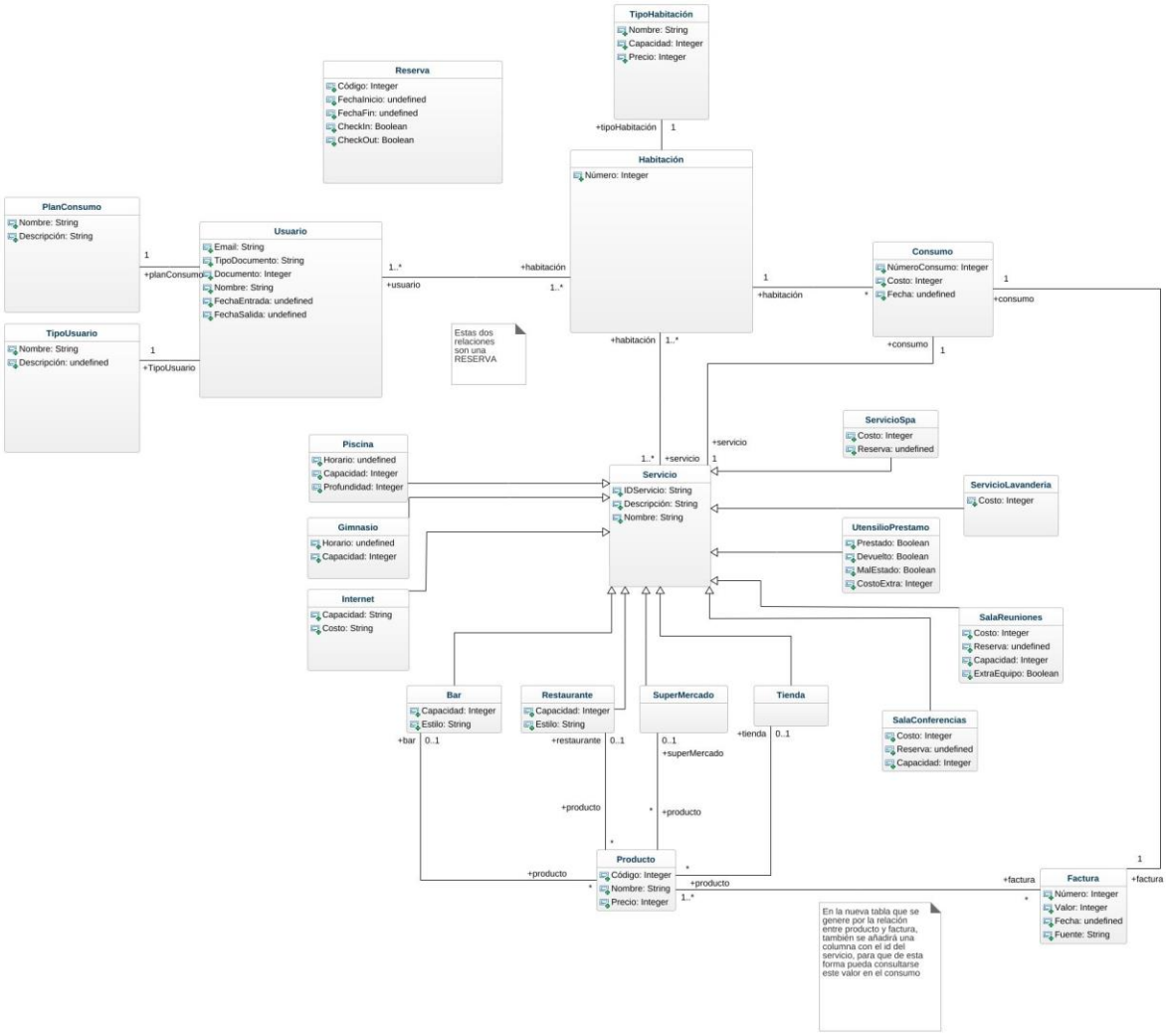
Luis Felipe Plazas – 202013155 - l.plazasp

David Almanza -202011293 – d.almanza

Nicolás Lara Gómez – 202012455 – n.lara

- **Análisis**

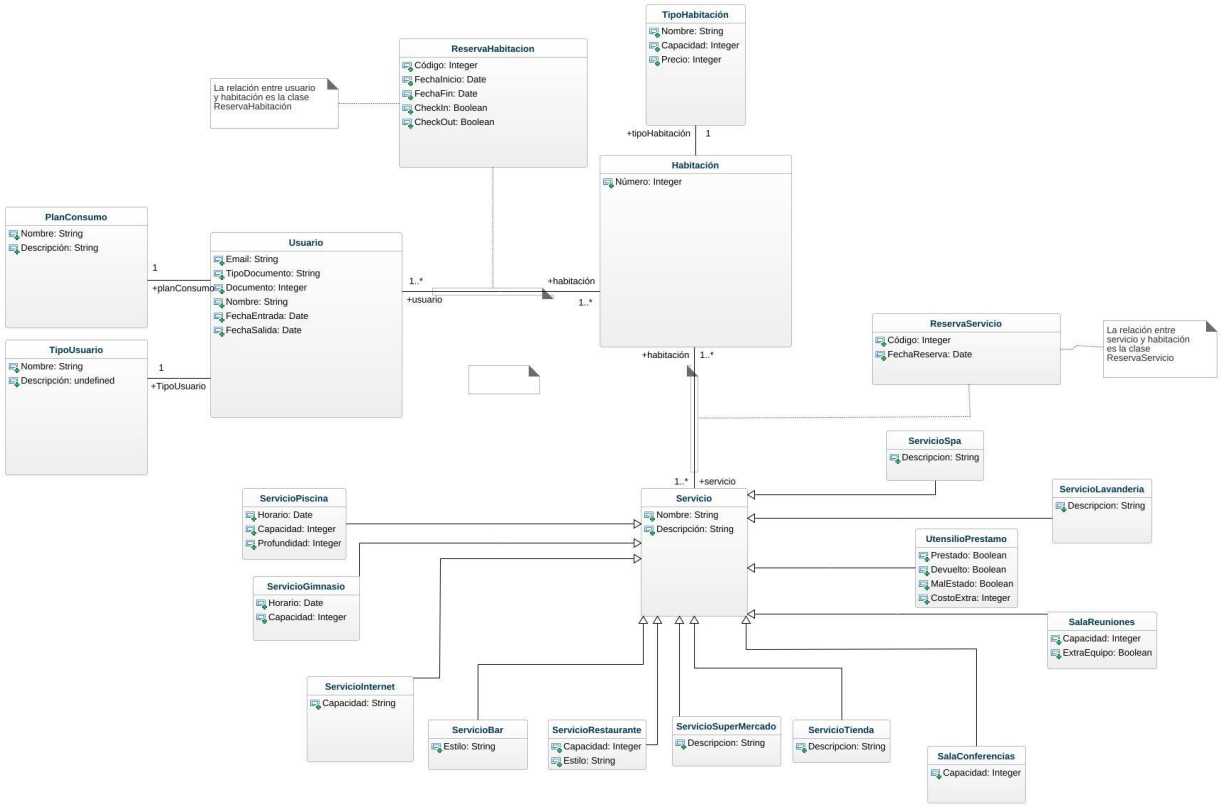
Diagrama de clases para la primera entrega:



A partir del diseño existente, se realizaron diferentes cambios al diagrama de clases previamente propuesto para la entrega de la iteración 1 del proyecto. Con los nuevos requerimientos que nos solicitan para esta entrega, se concluyó que la clase de SERVICIO iba a ser utilizada en demasía con respecto a las otras clases, y que no estábamos definiendo bien como íbamos a acceder a las clases que heredaban de esta, ya que teníamos como llave primaria un id que no nos proveía la información necesaria para ello. Es por esto por lo que se quitó el atributo ServicioID, y se dejó el nombre del servicio como la llave primaria de esta clase, y de todas las clases que heredan de ella. También se decidió quitar las clases PRODUCTO, FACTURA Y CONSUMO ya que para los requerimientos que se pidieron para esta entrega no los veíamos necesarios. Hay algunos

requerimientos que piden el consumo de ciertas personas y/o habitaciones, es por esto por lo que se decidió dejar el costo del servicio en la clase SERVICIO, y quitarle esos atributos a las clases que heredan de esta, de esta forma se puede acceder de manera más sencilla a este precio. Cabe aclarar que se cambiaron algunos nombres de las clases que heredaban de servicio, para saber que eran como tal un servicio. También se corrigieron ciertos problemas del diagrama, como que había algunos atributos que eran de tipo undefinded, y que algunas clases no tenían atributos, o no estaban relacionadas a nada dentro del diagrama.

Diagrama de clases para la segunda entrega:



- **Diseño de la aplicación**

1. Índices

1.

- Sí es necesario crear un índice.
- El índice creado será simple sobre la columna “Nombre” de la tabla “Servicios”. Debido a que este campo es la PK de dicha tabla, se garantiza unicidad de los registros, por tanto, su selectividad es del 100%.
- Debido a que no se están manejando valores numéricos en las consultas, no resulta útil emplear un índice Hash, motivo por el cual decidimos escoger un índice B+ primario (primario ya que estamos en una consulta que busca registros específicos basados en la clave primaria).

2.

- Sí es necesario crear un índice.
- Bajo la misma justificación del punto anterior, la selectividad del campo “Nombre” de la tabla “Servicios” es del 100%, motivo por el cual resulta muy útil aplicar un índice simple sobre esta columna. Para este caso no se buscan crear más índices ya que esto entorpecería mucho la eficiencia de las operaciones del CRUD.
- Debido a que no se están manejando valores numéricos en las consultas, no resulta útil emplear un índice Hash, motivo por el cual decidimos escoger un índice B+ primario.

3.

- Sí es necesario crear un índice.
- En este caso se va a crear nuevamente sobre un campo que es llave primaria de una relación (selectividad del 100%). Ahora va a ser sobre la columna “Numero” de la tabla “Habitaciones”. Para este caso, decidimos hacerlo exclusivamente sobre este campo y no sobre otros cuya selectividad también era muy alta (como “fecha_fin” o “fecha_inicio” de la tabla “reserva_habitacion") debido a que esto podría atentar contra la eficiencia de las demás funcionalidades de la aplicación.
- Debido a que no se están manejando valores numéricos en las consultas, no resulta útil emplear un índice Hash, motivo por el cual decidimos escoger un índice B+ primario.

4.

- Sí es necesario crear un índice.
- Para este RF se emplea el mismo índice sobre el campo “Nombre” de la relación “Servicios”. La justificación es la misma de los RF anteriores.
- En este caso sí se emplean valores numéricos dependiendo de la característica del servicio que se esté ingresando. Sin embargo, estos valores se manejan por rangos (es decir, con la instrucción BETWEEN en la consulta SQL), y, además, el índice está creado únicamente sobre el campo que no contiene valores numéricos, por tanto, resulta más útil emplear un índice B+.

5.

- Sí es necesario crear un índice.
- Para este RF se emplea el mismo índice sobre el campo “Nombre” de la relación “Servicios”. La justificación es la misma de los RF anteriores.
- Debido a que no se están manejando valores numéricos en las consultas (lo más cercano a esto son las fechas, pero aun así no es buena idea emplear un índice hash), no resulta útil emplear un índice Hash, motivo por el cual decidimos escoger un índice B+ primario.

6.

- Sí es necesario crear un índice.
- En este caso se emplea un índice sobre el campo de “fecha_inicio” de la relación “reserva_habitacion” debido a que, aunque varias reservas se pueden iniciar en el mismo día, la cantidad de datos tan grande hace que esto suceda muy poco y que la mayoría de las fechas de inicio sean diferentes, motivo por el cual su selectividad es lo suficientemente alta como para incluir un índice. Se emplearía otro índice en el campo de “fecha_reserva” de la relación “reserva_servicio” bajo la misma argumentación.
- Debido a que no se están manejando valores numéricos en las consultas, no resulta útil emplear un índice Hash, motivo por el cual decidimos escoger un índice B+ primario.

7.

- Sí es necesario crear un índice.
- En este caso se emplea un índice sobre el campo de “numero_documento” de la relación “usuarios” debido a que es la PK de esta tabla y su selectividad es del 100%, motivo por el cual resulta útil asignarle un índice para agilizar las consultas.
- Debido a que no se están manejando valores numéricos en las consultas, no resulta útil emplear un índice Hash, motivo por el cual decidimos escoger un índice B+ primario.

8.

- Sí es necesario crear un índice.
- Para este RF se emplea el mismo índice sobre el campo “Nombre” de la relación “Servicios”. La justificación es la misma de los RF anteriores.
- Debido a que no se están manejando valores numéricos en las consultas, no resulta útil emplear un índice Hash, motivo por el cual decidimos escoger un índice B+ primario.

9.

- Sí es necesario crear un índice.
- En este caso se emplea un índice sobre el campo de “numero_documento” de la relación “usuarios” debido a que es la PK de esta tabla y su selectividad es del 100%, motivo por el cual resulta útil asignarle un índice para agilizar las consultas.
- Debido a que no se están manejando valores numéricos en las consultas, no resulta útil emplear un índice Hash, motivo por el cual decidimos escoger un índice B+ primario.

10.

- Sí es necesario crear un índice.
- En este caso se emplea un índice sobre el campo de “numero_documento” de la relación “usuarios” debido a que es la PK de esta tabla y su selectividad es del 100%, motivo por el cual resulta útil asignarle un índice para agilizar las consultas.
- Debido a que no se están manejando valores numéricos en las consultas, no resulta útil emplear un índice Hash, motivo por el cual decidimos escoger un índice B+ primario.

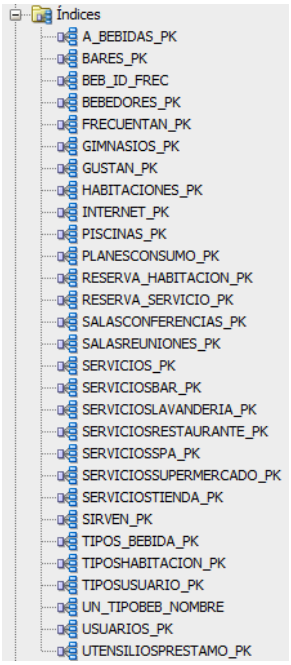
11.

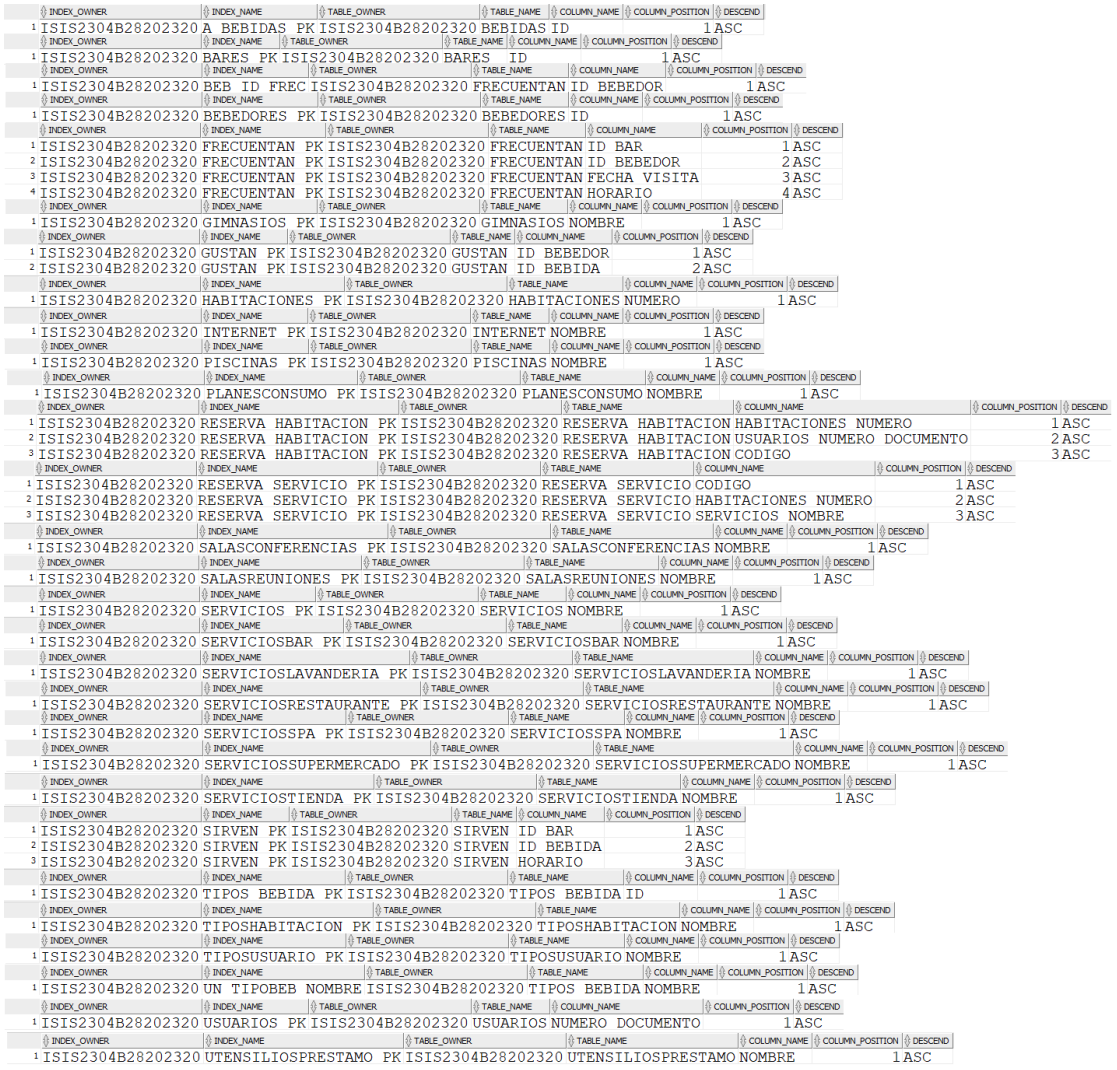
- Sí es necesario crear un índice.
- Para este RF se emplea el mismo índice sobre el campo “Nombre” de la relación “Servicios”. La justificación es la misma de los RF anteriores. Además, el uso de este en el INNER JOIN agiliza aún más la extracción de la información.
- Debido a que no se están manejando valores numéricos en las consultas, no resulta útil emplear un índice Hash, motivo por el cual decidimos escoger un índice B+ primario.

12.

- Sí es necesario crear un índice.
- En este caso se emplea un índice sobre el campo de “numero_documento” de la relación “usuarios” debido a que es la PK de esta tabla y su selectividad es del 100%, motivo por el cual resulta útil asignarle un índice para agilizar las consultas.
- Debido a que no se están manejando valores numéricos en las consultas, no resulta útil emplear un índice Hash, motivo por el cual decidimos escoger un índice B+ primario.

A continuación, se muestran todos los índices generados automáticamente. Además, se incluye la información generada por Oracle.





	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	A_BEBIDAS_PK	ISIS2304B28202320	BEBIDAS	ID	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	BARES_PK	ISIS2304B28202320	BARES	ID	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	BEB_ID_FREC	ISIS2304B28202320	FRECUENTAN	ID_BEBEDOR	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	BEBEDORES_PK	ISIS2304B28202320	BEBEDORES	ID	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	FRECUENTAN_PK	ISIS2304B28202320	FRECUENTAN	ID_BAR	1	ASC
2	ISIS2304B28202320	FRECUENTAN_PK	ISIS2304B28202320	FRECUENTAN	ID_BEBEDOR	2	ASC
3	ISIS2304B28202320	FRECUENTAN_PK	ISIS2304B28202320	FRECUENTAN	FECHA_VISITA	3	ASC
4	ISIS2304B28202320	FRECUENTAN_PK	ISIS2304B28202320	FRECUENTAN	HORARIO	4	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	GIMNASIOS_PK	ISIS2304B28202320	GIMNASIOS	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	GUSTAN_PK	ISIS2304B28202320	GUSTAN	ID_BEBEDOR	1	ASC
2	ISIS2304B28202320	GUSTAN_PK	ISIS2304B28202320	GUSTAN	ID_BEBIDA	2	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	HABITACIONES_PK	ISIS2304B28202320	HABITACIONES	NUMERO	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	INTERNET_PK	ISIS2304B28202320	INTERNET	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	PISCINAS_PK	ISIS2304B28202320	PISCINAS	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	PLANESCONSUMO_PK	ISIS2304B28202320	PLANESCONSUMO	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	RESERVA_HABITACION_PK	ISIS2304B28202320	RESERVA_HABITACION	HABITACIONES_NUMERO	1	ASC
2	ISIS2304B28202320	RESERVA_HABITACION_PK	ISIS2304B28202320	RESERVA_HABITACION	USUARIOS_NUMERO_DOCUMENTO	2	ASC
3	ISIS2304B28202320	RESERVA_HABITACION_PK	ISIS2304B28202320	RESERVA_HABITACION	CODIGO	3	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	RESERVA_SERVICIO_PK	ISIS2304B28202320	RESERVA_SERVICIO	CODIGO	1	ASC
2	ISIS2304B28202320	RESERVA_SERVICIO_PK	ISIS2304B28202320	RESERVA_SERVICIO	HABITACIONES_NUMERO	2	ASC
3	ISIS2304B28202320	RESERVA_SERVICIO_PK	ISIS2304B28202320	RESERVA_SERVICIO	SERVICIOS_NOMBRE	3	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	SALASCONFERENCIAS_PK	ISIS2304B28202320	SALASCONFERENCIAS	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	SALASREUNIONES_PK	ISIS2304B28202320	SALASREUNIONES	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	SERVICIOS_PK	ISIS2304B28202320	SERVICIOS	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	SERVICIOSBAR_PK	ISIS2304B28202320	SERVICIOSBAR	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	SERVICIOSLAVANDERIA_PK	ISIS2304B28202320	SERVICIOSLAVANDERIA	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	SERVICIOSRESTAURANTE_PK	ISIS2304B28202320	SERVICIOSRESTAURANTE	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	SERVICIOSSPA_PK	ISIS2304B28202320	SERVICIOSSPA	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	SERVICIOSSUPERMERCADO_PK	ISIS2304B28202320	SERVICIOSSUPERMERCADO	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	SERVICIOSTIENDA_PK	ISIS2304B28202320	SERVICIOSTIENDA	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	SIRVEN_PK	ISIS2304B28202320	SIRVEN	ID_BAR	1	ASC
2	ISIS2304B28202320	SIRVEN_PK	ISIS2304B28202320	SIRVEN	ID_BEBIDA	2	ASC
3	ISIS2304B28202320	SIRVEN_PK	ISIS2304B28202320	SIRVEN	HORARIO	3	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	TIPOS_BEBIDA_PK	ISIS2304B28202320	TIPOS_BEBIDA	ID	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	TIPOSHABITACION_PK	ISIS2304B28202320	TIPOSHABITACION	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	TIPOSUSUARIO_PK	ISIS2304B28202320	TIPOSUSUARIO	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	UN_TIPOBEB_NOMBRE	ISIS2304B28202320	TIPOS_BEBIDA	NOMBRE	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	USUARIOS_PK	ISIS2304B28202320	USUARIOS	NUMERO_DOCUMENTO	1	ASC

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304B28202320	UTENSILIOSPRESTAMO_PK	ISIS2304B28202320	UTENSILIOSPRESTAMO	NOMBRE	1	ASC

Toda la información brindada por Oracle consta de lo siguiente: Dueño del índice (el cual concuerda con el usuario que estamos empleando para acceder a la base de datos), nombre del índice (escogido por Oracle), dueño de la tabla (igual al dueño del indice), nombre de la tabla, nombre de la columna(s) escogida(s) para crear el índice, posición(es) de la(s) columna(s) y el orden (ASC).

Los criterios que sigue Oracle (y que siguió para nuestras tablas) para crear estos índices automáticamente son los siguientes:

- Integridad: Oracle crea índices automáticamente para garantizar la integridad de los datos. Por ejemplo, un índice único se crea automáticamente en una columna declarada como PK (i.e. servicios.nombre).
- Claves foráneas: Se crean índices en las columnas que participan en relaciones de clave foránea para mejorar el rendimiento de las consultas que involucran joins entre tablas relacionadas.

- Columnas especiales: Oracle sugiere la creación de índices en columnas que se consultan con frecuencia en declaraciones SQL o en aquellas que filtran datos de manera efectiva.
- Ordenación. Oracle sugiere la creación de índices en columnas utilizadas en cláusulas "ORDER BY" o "GROUP BY" para mejorar el rendimiento de las consultas de ordenación y agrupación.

En general, todos estos índices ayudan mucho a optimizar la eficiencia de las consultas SQL hechas para este proyecto.

2. Diseño de las consultas

- **Sentencias SQL**

- El archivo denominado ConsultasP2.sql contiene las consultas SQL hechas para cada uno de los RF del proyecto.

- **Análisis de planes de ejecución de los requerimientos**

1. RF1:

Para el RF1, se está pidiendo el consumo de cada habitación a lo largo de un año.

Caso 1: En nuestra carga de los datos se están utilizando 1000 habitaciones, por lo que, si no se especifica lo contrarios, la respuesta va a devolver 1000 registros.

Resultado:

	SUM(SERVICIOS.COSTO)	NUMERO
977	8936371	977
978	11108141	978
979	13329261	979
980	11056520	980
981	16997744	981
982	9405570	982
983	10891607	983
984	11259505	984
985	11931476	985
986	13702880	986
987	14743547	987
988	13699846	988
989	11319809	989
990	13182599	990
991	14371921	991
992	13582569	992
993	12899866	993
994	14591550	994
995	14086939	995
996	12295265	996
997	12972709	997
998	8957685	998
999	10590456	999
1000	8977237	1000

Registros retornados: 1000

Valores de los parámetros: No se especificó ningún parámetro

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1000	226
SORT		GROUP BY	1000	226
HASH JOIN			48752	224
Access Predicates				
SERVICIOS.NOMBRE=RESERVA_SERVICIO.SERVICIOS_NOMBRE				
TABLE ACCESS	SERVICIOS	FULL	188	4
TABLE ACCESS	RESERVA_SERVICIO	FULL	48752	219
Filter Predicates				
RESERVA_SERVICIO.FECHA_RESERVA>=SYSDATE@!-INTERVAL'+01-00' YEAR(2) TO MONTH				
Other XML				

Tiempo obtenido:

Todas las Filas Recuperadas: 1000 en 0,138 segundos

Caso 2: Ahora se pedirá que solo retorne las primeras 100 habitaciones

Resultado:

	SUM(SERVICIOS.COSTO)	NUMERO
77	14139737	77
78	16698180	78
79	11805118	79
80	10349471	80
81	14132637	81
82	13404844	82
83	13543258	83
84	12359929	84
85	13001311	85
86	14884085	86
87	12734379	87
88	15781709	88
89	12855724	89
90	16147020	90
91	12292042	91
92	12431123	92
93	15011982	93
94	9071090	94
95	11911364	95
96	15172227	96
97	13204072	97
98	17457178	98
99	11522810	99
100	14562866	100

Registros retornados: 100

Valores de los parámetros: Solo las primeras 100 habitaciones

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			100	226
VIEW	SYS.null		100	226
Filter Predicates				
from$_subquery$_006.rowlimit_$$_rownumber<=100				
WINDOW		NOSORT STOPKEY	1000	226
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY RESERVA_SERVICIO.HABITACIONES_NUMERO)<=100				
SORT		GROUP BY	1000	226
HASH JOIN			48752	224
Access Predicates				
SERVICIOS.NOMBRE=RESERVA_SERVICIO.SERVICIOS_NOMBRE				
TABLE ACCESS	SERVICIOS	FULL	188	4
TABLE ACCESS	RESERVA_SERVICIO	FULL	48752	219
Filter Predicates				
RESERVA_SERVICIO.FECHA_RESERVA>=SYSDATE@!-INTERVAL'+01-00' YEAR(2) TO MONTH				

Tiempo obtenido:

Todas las Filas Recuperadas: 100 en 0,054 segundos

2. RF2:

Para el RF2, se quiere saber cuales fueron los 20 servicios más populares en un periodo de tiempo dado, para esto se puede jugar con los intervalos de tiempo en los que se pide la consulta.

Caso 1: Para el primer ejemplo se puso como rango de tiempo entre el 17 de mayo y 18 de mayo, para así saber cual fue el servicio más utilizado en esos días:

Resultado:

	NOMBRE	NUMERO_RESERVAS
1	Steam Room	5
2	tie clip	5
3	Deep Tissue Massage	4
4	apples	4
5	tequila	4
6	soda	4
7	Fish Tacos	4
8	yogurt	4
9	Margherita Pizza	3
10	Sangria	3
11	Chicken Satay	3
12	Cosmopolitan	3
13	nose stud	3
14	bananas	3
15	Sushi Sashimi Platter	3
16	Vegetable Curry	3
17	pasta	3
18	waist belt	3
19	body chain	3

Registros retornados: 20

Valores de los parámetros: Entre el 17 y el 18 de mayo

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			20	220
SORT		ORDER BY	20	220
VIEW	SYS.null		20	219
Filter Predicates				
from$_subquery$_004.rowlimit_$$_rownumber<=20				
WINDOW		SORT PUSHED RANK	164	219
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC)<=20				
HASH		GROUP BY	164	219
FILTER				
Filter Predicates				
TO_DATE('18/MAY/2023')>=TO_DATE('17/MAY/2023')				
TABLE ACCESS	RESERVA_SERVICIO	FULL	404	217
Filter Predicates				
AND				
RESERVA_SERVICIO.FECHA_RESERVA>='17/MAY/2023'				
RESERVA_SERVICIO.FECHA_RESERVA<='18/MAY/2023'				

Tiempo obtenido:

Todas las Filas Recuperadas: 20 en 0,029 segundos

Caso 2: Ahora, se va a realizar la misma consulta, pero con un rango de tiempo mayor, entre el 1 de enero y 31 de octubre, el resultado fue el siguiente:

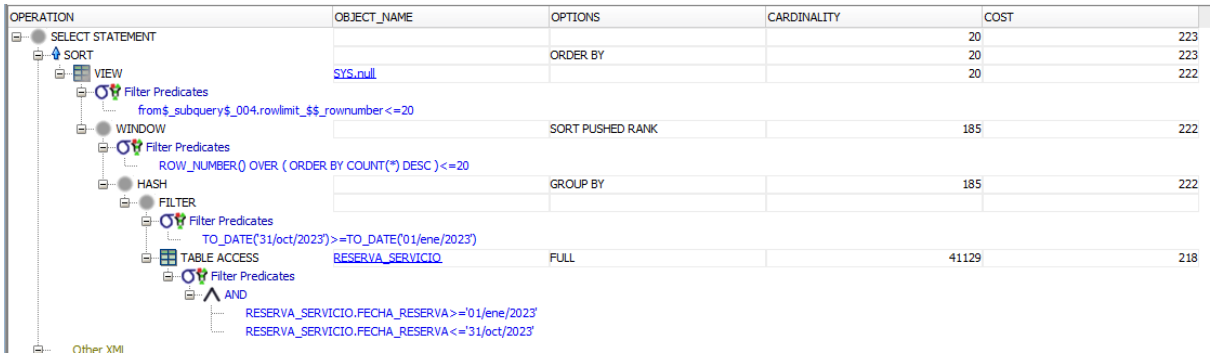
Resultado:

	NOMBRE	NUMERO_RESERVAS
1	necklace	442
2	Sala de conferencia	256
3	Beef Teriyaki	255
4	beer	254
5	champagne	254
6	Gin and Tonic	250
7	Chicken Alfredo	249
8	Manhattan	248
9	Internet	248
10	Chicken Parmesan	247
11	pasta	247
12	Chicken Quesadilla	247
13	body piercing	246
14	pendant	246
15	Margherita Flatbread	244
16	Margherita Pizza	243
17	juice	241
18	Vegetable Stir Fry	241
19	Body Scrub	241
20	Vegetable Curry	240

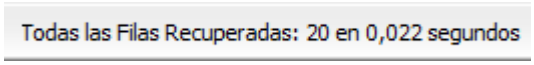
Registros retornados: 20

Valores de los parámetros: Entre el 1 de enero y el 31 de octubre

Plan de consulta:



Tiempo obtenido:



Se puede evidenciar que el número de reservas aumentó bastante, ya que el rango de tiempo que se especificó fue mucho mayor.

3. RF3:

Para el RF3, se está pidiendo el índice de ocupación de las habitaciones a lo largo de un año.

Caso 1: En nuestra carga de los datos se están utilizando 1000 habitaciones, por lo que, si no se especifica lo contrarios, la respuesta va a devolver 1000 registros.

Resultado:

	NUMERO_HABITACION	TOTAL_DIAS_OCUPADA	INDICE_OCUPACION
977	977	3704	101
978	978	3292	90
979	979	3345	92
980	980	3039	83
981	981	3271	90
982	982	3564	98
983	983	3749	103
984	984	3426	94
985	985	3050	84
986	986	3353	92
987	987	3570	98
988	988	3667	100
989	989	3414	94
990	990	3026	83
991	991	3784	104
992	992	3407	93
993	993	3237	89
994	994	3373	92
995	995	2864	78
996	996	3449	94
997	997	3181	87
998	998	3051	84
999	999	3165	87
1000	1000	3136	86

Registros retornados: 1000

Valores de los parámetros: No se especificó ningún parámetro

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1000	226
SORT		GROUP BY	1000	226
TABLE ACCESS	RESERVA_HABITACION	FULL	213631	219
Filter Predicates				
RESERVA_HABITACION.FECHA_INICIO>=SYSDATE@!-INTERVAL'+01-00' YEAR(2) TO MONTH				

Tiempo obtenido:

Todas las Filas Recuperadas: 1000 en 0,195 segundos

Caso 2: Ahora se pedirá que solo retorne las primeras 100 habitaciones con menor índice de ocupación

Resultado:

	NUMERO_HABITACION	TOTAL_DIAS_OCUPADA	INDICE_OCUPACION
79	248	2963	81
80	277	2962	81
81	569	2971	81
82	938	2954	81
83	26	2970	81
84	766	2959	81
85	309	2954	81
86	288	2947	81
87	646	2958	81
88	297	2964	81
89	614	2966	81
90	468	2974	81
91	967	2972	81
92	262	2940	81
93	162	2945	81
94	42	2973	81
95	660	2961	81
96	647	2962	81
97	798	2943	81
98	463	2981	82
99	64	2994	82
100	27	3004	82

Registros retornados: 100

Valores de los parámetros: Solo las primeras 100 habitaciones con menor índice de ocupación

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			100	233
SORT		ORDER BY	100	233
VIEW	SYS.null		100	232
Filter Predicates				
from$_subquery$_004.rowlimit_$$_rownumber<=100				
WINDOW		SORT PUSHED RANK	1000	232
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY ROUND(SUM(RESERVA_HABITACION.FECHA_FIN-RESERVA_HABITACION.FECHA_INICIO)/365*10))<=100				
HASH		GROUP BY	1000	232
TABLE ACCESS	RESERVA_HABITACION	FULL	213504	219
Filter Predicates				
RESERVA_HABITACION.FECHA_INICIO>=SYSDATE@!-INTERVAL'+01-00' YEAR(2) TO MONTH				

Tiempo obtenido:

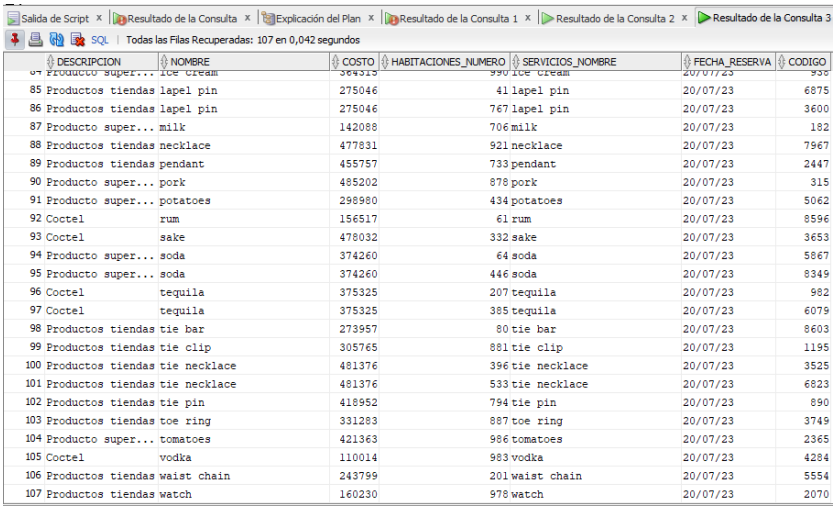
Todas las Filas Recuperadas: 100 en 0,066 segundos

4. RF4

Para el RF4 se piden los servicios que estén en un rango determinado, ya sea de tiempo, de precio, de cierto empleado que los registro o con varias características en simultáneo.

Caso 1: Primero probaremos que servicios se registraron en el día 20 de julio, y que tengan un precio entre 100000 y 500000, el resultado es el siguiente

Resultado:



Salida de Script | Resultado de la Consulta | Explicación del Plan | Resultado de la Consulta 1 | Resultado de la Consulta 2 | Resultado de la Consulta 3

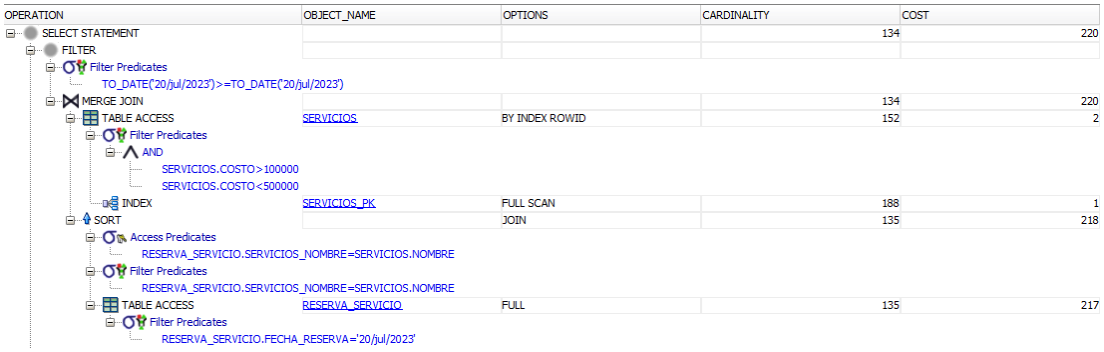
Todas las Filas Recuperadas: 107 en 0,042 segundos

	DESCRIPCION	NOMBRE	COSTO	HABITACIONES_NUMERO	SERVICIOS_NOMBRE	FECHA_RESERVA	CODIGO
84	Producto super...	ice cream	364313	990	ice cream	20/07/23	956
85	Productos tiendas	lapel pin	275046	41	lapel pin	20/07/23	6875
86	Productos tiendas	lapel pin	275046	767	lapel pin	20/07/23	3600
87	Producto super...	milk	142088	706	milk	20/07/23	182
88	Productos tiendas	necklace	477831	921	necklace	20/07/23	7967
89	Productos tiendas	pendant	455757	733	pendant	20/07/23	2447
90	Producto super...	pork	485202	878	pork	20/07/23	315
91	Producto super...	potatoes	298980	434	potatoes	20/07/23	5062
92	Coctel	rum	156517	61	rum	20/07/23	8596
93	Coctel	sake	478032	332	sake	20/07/23	3653
94	Producto super...	soda	374260	64	soda	20/07/23	5867
95	Producto super...	soda	374260	446	soda	20/07/23	8349
96	Coctel	tequila	375325	207	tequila	20/07/23	982
97	Coctel	tequila	375325	385	tequila	20/07/23	6079
98	Productos tiendas	tie bar	273957	80	tie bar	20/07/23	8603
99	Productos tiendas	tie clip	305765	881	tie clip	20/07/23	1195
100	Productos tiendas	tie necklace	481376	396	tie necklace	20/07/23	3525
101	Productos tiendas	tie necklace	481376	533	tie necklace	20/07/23	6823
102	Productos tiendas	tie pin	418952	794	tie pin	20/07/23	890
103	Productos tiendas	toe ring	331283	887	toe ring	20/07/23	3749
104	Producto super...	tomatoes	421363	986	tomatoes	20/07/23	2365
105	Coctel	vodka	110014	983	vodka	20/07/23	4284
106	Productos tiendas	waist chain	243799	201	waist chain	20/07/23	5554
107	Productos tiendas	watch	160230	978	watch	20/07/23	2070

Registros retornados: 107

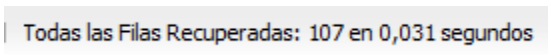
Valores de los parámetros: El día 20 de julio, precio entre 100000 y 500000

Plan de consulta:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			134	220
FILTER				
Filter Predicates				
TO_DATE('20/jul/2023')>=TO_DATE('20/jul/2023')				
MERGE JOIN			134	220
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	152	2
Filter Predicates				
AND				
SERVICIOS.COSTO>100000				
SERVICIOS.COSTO<500000				
INDEX	SERVICIOS_PK	FULL SCAN	188	1
SORT		JOIN	135	218
Access Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
Filter Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
TABLE ACCESS	RESERVA_SERVICIO	FULL	135	217
Filter Predicates				
RESERVA_SERVICIO.FECHA_RESERVA='20/jul/2023'				

Tiempo obtenido:



Todas las Filas Recuperadas: 107 en 0,031 segundos

Caso 2: Ahora, se va a probar con los meses de junio y julio, y sin ninguna restricción de precio a ver cuál es el número de registros que retorna:

Resultado:

	DESCRIPCION	NOMBRE	COSTO	HABITACIONES_NUMERO	SERVICIOS_NOMBRE	FECHA_RESERVA	CODIGO
8333	Producto ...	yogurt	174035	315	yogurt	25/06/23	8675
8334	Producto ...	yogurt	174035	747	yogurt	26/07/23	1085
8335	Producto ...	yogurt	174035	613	yogurt	09/07/23	302
8336	Producto ...	yogurt	174035	875	yogurt	09/07/23	4521
8337	Producto ...	yogurt	174035	13	yogurt	01/07/23	4003
8338	Producto ...	yogurt	174035	286	yogurt	14/06/23	39
8339	Producto ...	yogurt	174035	954	yogurt	21/06/23	7756
8340	Producto ...	yogurt	174035	498	yogurt	18/07/23	2015
8341	Producto ...	yogurt	174035	202	yogurt	10/07/23	5577
8342	Producto ...	yogurt	174035	435	yogurt	24/06/23	2867
8343	Producto ...	yogurt	174035	315	yogurt	14/07/23	9314
8344	Producto ...	yogurt	174035	101	yogurt	02/06/23	5715
8345	Producto ...	yogurt	174035	367	yogurt	02/06/23	2319
8346	Producto ...	yogurt	174035	85	yogurt	18/07/23	2106
8347	Producto ...	yogurt	174035	829	yogurt	07/06/23	7092
8348	Producto ...	yogurt	174035	848	yogurt	25/06/23	286
8349	Producto ...	yogurt	174035	988	yogurt	13/06/23	6503
8350	Producto ...	yogurt	174035	123	yogurt	19/06/23	8633
8351	Producto ...	yogurt	174035	798	yogurt	09/07/23	4760
8352	Producto ...	yogurt	174035	197	yogurt	15/07/23	41
8353	Producto ...	yogurt	174035	221	yogurt	04/06/23	1619
8354	Producto ...	yogurt	174035	722	yogurt	25/07/23	1475
8355	Producto ...	yogurt	174035	809	yogurt	28/06/23	7993

Registros retornados: 8355

Valores de los parámetros: Los meses de junio y julio

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			8361	221
FILTER				
Filter Predicates				
TO_DATE('31/jul/2023')>=TO_DATE('01/jun/2023')				
MERGE JOIN			8361	221
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	188	2
INDEX	SERVICIOS_PK	FULL SCAN	188	1
SORT		JOIN	8361	219
Access Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
Filter Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
TABLE ACCESS	RESERVA_SERVICIO	FULL	8361	217
Filter Predicates				
AND				
RESERVA_SERVICIO.FECHA_RESERVA>='01/jun/2023'				
RESERVA_SERVICIO.FECHA_RESERVA<='31/jul/2023'				

Tiempo obtenido:

Todas las Filas Recuperadas: 8355 en 0,868 segundos

Se observa que esta cantidad de registros es muchísimo mayor a la vista anteriormente, ya que el rango de tiempo es mucho mayor, y no hay ninguna restricción de precio puesta en la consulta.

5. RF5

Para el RF5 se pide el consumo de un usuario dado, en un rango de fechas determinado.

Caso 1: Primero probaremos con el usuario con documento 1, y con el rango de fechas entre mayo 17 y junio 17

Resultado:

	NOMBRE	COSTO
134	vodka	110014
135	waistcoat	399639
136	watch chain	134674
137	BBQ Ribs	223383
138	Chicken Tikka Masala	80579
139	Grilled Salmon	415910
140	Hair Styling	141946
141	Manicure	67578
142	Moscow Mule	168125
143	Old Fashioned	124125
144	Peking Duck	232933
145	Shrimp Scampi	253288
146	Sushi Rolls	197500
147	Tandoori Chicken	249114
148	beer	374747
149	belly button ring	125039
150	hairpin	42618
151	juice	168591
152	lapel pin	275046
153	pork	485202
154	sake	478032
155	soda	374260
156	waist belt	171283
157	waist chain	243799

Registros retornados: 157

Valores de los parámetros: El usuario con doc 1, y el rango de tiempo es entre mayo 17 y junio 17

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			463	409
HASH		UNIQUE	463	409
FILTER				
Filter Predicates				
TO_DATE('17/jun/2023')>=TO_DATE('17/may/2023')				
HASH JOIN		RIGHT SEMI	463	408
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
INDEX	RESERVA_HABITACION_PK	FAST FULL SCAN	108	187
Filter Predicates				
RESERVA_HABITACION.USUARIOS_NUMERO_DOCUMENTO=1				
MERGE JOIN			4450	221
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	188	2
INDEX	SERVICIOS_PK	FULL SCAN	188	1
SORT		JOIN	4450	219
Access Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
Filter Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
TABLE ACCESS	RESERVA_SERVICIO	FULL	4450	217
Filter Predicates				
AND				
RESERVA_SERVICIO.FECHA_RESERVA>='17/may/2023'				
RESERVA_SERVICIO.FECHA_RESERVA<='17/jun/2023'				

Tiempo obtenido:

Todas las Filas Recuperadas: 157 en 0,056 segundos

Caso 2: Ahora, se va a probar el usuario con id 4271 el día 30 de junio

Resultado:

	NOMBRE	COSTO
1	Body Wrap	109389
2	Cosmopolitan	195619
3	Mojito	492668
4	frozen pizza	368241
5	Margherita Flatbread	413297
6	Piscina	118813
7	cookies	482032
8	cufflinks	149133
9	rice	287259
10	Massage	185261
11	cereal	164423
12	Hamburger	360355
13	ankle socks	396863

Registros retornados: 13

Valores de los parámetros: El usuario con doc 4271 y el día 30 de junio

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			109	356
HASH		UNIQUE	109	356
FILTER				
Filter Predicates				
TO_DATE('30/jun/2023')>=TO_DATE('30/jun/2023')				
HASH JOIN		SEMI	109	355
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
NESTED LOOPS		SEMI	109	355
STATISTICS COLLECTOR				
MERGE JOIN			135	220
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	188	2
INDEX	SERVICIOS_PK	FULL SCAN	188	1
SORT		JOIN	135	218
Access Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
Filter Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
TABLE ACCESS	RESERVA_SERVICIO	FULL	135	217
Filter Predicates				
RESERVA_SERVICIO.FECHA_RESERVA='30/jun/2023'				
INDEX	RESERVA_HABITACION_PK	RANGE SCAN	88	1
Access Predicates				
AND				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
RESERVA_HABITACION.USUARIOS_NUMERO_DOCUMENTO=4271				
INDEX	RESERVA_HABITACION_PK	FAST FULL SCAN	88	1
Filter Predicates				
RESERVA_HABITACION.USUARIOS_NUMERO_DOCUMENTO=4271				

Tiempo obtenido:

Todas las Filas Recuperadas: 13 en 0,028 segundos

6. RF6

Para el RF6 se pide indicar cuáles fueron las fechas de los días de mayor ocupación (mayor cantidad habitaciones ocupadas), las fechas de mayores ingresos (mayor cantidad de consumos realizados) y también las fechas de menor demanda (menor ocupación).

Caso 1: Primero se consultará los 10 días de mayor ocupación

Resultado:

	FECHA_INICIO	OCUPACION
1	09/08/23	660
2	19/07/23	658
3	05/12/22	654
4	05/06/23	649
5	22/11/22	648
6	06/05/23	647
7	02/04/23	647
8	15/08/23	643
9	19/03/23	639
10	02/06/23	639

Registros retornados: 10

Valores de los parámetros: Primeros 10 días con mayor ocupación

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			10	229
SORT		ORDER BY	10	229
VIEW	SYS.null		10	228
Filter Predicates				
from$_subquery$_002.rowlimit_$$_rownumber<=10				
WINDOW		SORT PUSHED RANK	365	228
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC)<=10				
HASH		GROUP BY	365	228
TABLE ACCESS	RESERVA_HABITACION	FULL	215095	215

Tiempo obtenido:

Todas las Filas Recuperadas: 10 en 0,024 segundos

Caso 2: Ahora, los 50 días con mayores ingresos en el hotel

Resultado:

	FECHA_RESERVA	INGRESOS
25	03/01/21	44133515
26	15/10/21	44094303
27	11/07/19	44067481
28	11/05/22	43882738
29	29/12/21	43874405
30	29/04/21	43871096
31	04/10/21	43850098
32	03/08/19	43814518
33	16/03/21	43775906
34	01/12/20	43767326
35	23/09/20	43677704
36	15/01/23	43621525
37	05/04/19	43612379
38	21/05/23	43591118
39	09/07/20	43455259
40	31/03/23	43445273
41	04/05/20	43430443
42	03/12/18	43411950
43	06/09/20	43359335
44	01/09/19	43262991
45	24/10/21	43134718
46	12/03/22	43123474
47	09/12/18	43097090
48	05/08/20	43068400
49	19/11/22	43063629
50	06/10/21	43023604

Registros retornados: 50

Valores de los parámetros: 50 días con mayores ingresos

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			50	236
SORT		ORDER BY	50	236
VIEW	SYS.null		50	235
Filter Predicates				
from$_subquery$_004.rowlimit_$$_rownumber<=50				
WINDOW		SORT PUSHED RANK	1825	235
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY SUM(SERVICIOS.COSTO) DESC)<=50				
HASH		GROUP BY	1825	235
HASH JOIN			245965	220
Access Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
TABLE ACCESS	SERVICIOS	FULL	188	4
TABLE ACCESS	RESERVA_SERVICIO	FULL	245965	215

Tiempo obtenido:

Se han recuperado 50 filas en 0,048 segundos

Caso 3: Ahora, los 200 días con menor ocupación

Resultado:

	FECHA_INICIO	OCUPACION
175	23/09/23	589
176	08/08/23	589
177	11/07/23	589
178	07/01/23	590
179	24/02/23	590
180	01/07/23	590
181	18/10/23	590
182	18/08/23	590
183	15/04/23	590
184	30/04/23	590
185	29/01/23	591
186	09/06/23	591
187	26/08/23	591
188	03/04/23	591
189	26/05/23	591
190	21/03/23	591
191	29/05/23	592
192	26/03/23	592
193	14/02/23	592
194	26/09/23	592
195	10/08/23	592
196	01/02/23	592
197	05/10/23	593
198	15/06/23	593
199	19/05/23	593
200	06/07/23	593

Registros retornados: 200

Valores de los parámetros: 200 días con menor ocupación

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			200	229
SORT		ORDER BY	200	229
VIEW	SYS.null		200	228
Filter Predicates				
from$_subquery$_002.rowlimit_$$_rownumber<=200				
WINDOW		SORT PUSHED RANK	365	228
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*))<=200				
HASH		GROUP BY	365	228
TABLE ACCESS	RESERVA_HABITACION	FULL	215095	215

Tiempo obtenido:

Todas las Filas Recuperadas: 200 en 0,044 segundos

Se pueden observar ciertas diferencias entre las diferentes consultas para el RF6, por ejemplo, las consultas que tienen que ver con la ocupación tienen el mismo costo, ya que están operando sobre los mismo datos, lo único que cambia es como se organizan.

7. RF7

Para el RF7 debe encontrar a los clientes que se consideran buenos clientes. Se considera buen cliente si ha estado en el hotel por lo menos dos semanas (no necesariamente en una sola estadía) o si ha consumido más de $15’000.000.oo, durante el último año de operación de HotelAndes.

Caso 1: Primero se consultará sin más a los buenos clientes

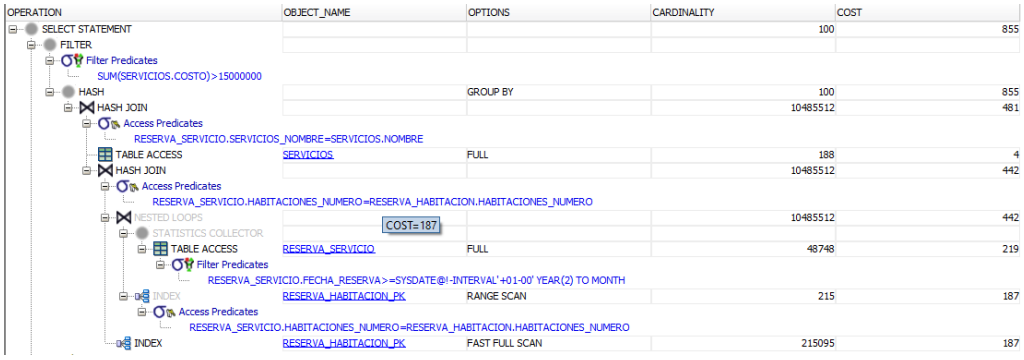
Resultado:

	NUMERO_DOCUMENTO	SUM(SERVICIOS.COSTO)
1960	4894	1130961283
1961	2253	1178478504
1962	826	1227950574
1963	91	1207253464
1964	1981	2030856248
1965	2270	1110279580
1966	803	1134798310
1967	138	944431269
1968	1769	1265966895
1969	1937	1098531931
1970	346	921002675
1971	899	962308716
1972	2470	1032627781
1973	2949	859800667
1974	3912	1180376259
1975	2268	1155706932
1976	676	1109116501
1977	616	1162312692
1978	3404	1164576958
1979	1064	958786275
1980	1494	1025342864
1981	2673	1118051223
1982	3148	1031798425
1983	2018	1162273484
1984	1136	1212953681
1985	2455	911075957

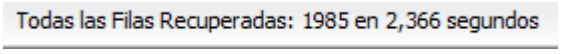
Registros retornados: 1985

Valores de los parámetros: Consumos mayor a 15 millones, estadía de más de 2 semanas

Plan de consulta:



Tiempo obtenido:



Caso 2: Ahora, se probará con valores mayores, aquellos clientes que hayna gastado más de 3000 millones de pesos

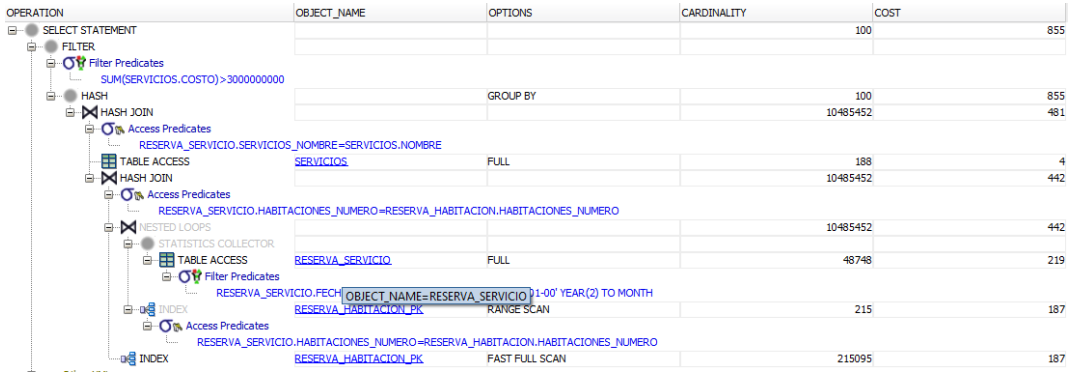
Resultado:

	NUMERO_DOCUMENTO	SUM(SERVICIOS.COSTO)
98	1546	3103185586
99	1882	4294816383
100	1489	5526781547
101	1208	3095901835
102	1997	4609673551
103	1439	3406011550
104	1251	3322319033
105	1341	4057366621
106	1098	3285881389
107	1177	4419579777
108	1314	4249165727
109	1532	4232934407
110	1640	3182194328
111	1836	3034592286
112	1366	3558088115
113	1662	6515037129
114	1781	4557707884
115	1454	3491503055
116	1706	3166439095
117	1817	3007488187
118	1455	3048361308
119	2954	3583412113
120	1163	3358995365
121	1550	3394087057
122	3183	3703699894
123	1581	3225205675

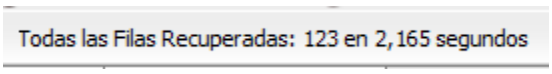
Registros retornados: 123

Valores de los parámetros: Clientes que han gastado más de 3000 millones de pesos

Plan de consulta:



Tiempo obtenido:



8. RF8

Para el RF8 se debe encontrar los servicios que hayan sido solicitados menos de 3 veces semanales, durante el último año de operación de HotelAndes.

Caso 1: Se intentará encontrar los servicios solicitados menos de 5 veces semanales

Resultado:

	NOMBRE_DEL_SERVICIO
66	tie dye shirt
67	ankle strap
68	Facial
69	Peking Duck
70	ring
71	tie bar
72	Mimosa
73	Waxing
74	watch
75	Caesar Wrap
76	Manicure
77	Fish and Chips
78	Caesar Salad
79	Pad Thai
80	tiara
81	Makeup Application
82	Lemon Herb Roast...
83	yogurt
84	Lobster Bisque
85	waist bag
86	watch chain
87	White Russian
88	Gimnasio

Registros retornados: 88

Valores de los parámetros: Servicios solicitados menos de 5 veces

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			10	515
FILTER				
Filter Predicates				
SUM(CANTIDAD_DE_SOLICITUDES)/COUNT(CANTIDAD_DE_SOLICITUDES)<5				
HASH		GROUP BY	10	515
VIEW			47355	515
HASH		GROUP BY	47355	515
TABLE ACCESS	RESERVA_SERVICIO	FULL	48815	223
Filter Predicates				
RS.FECHA_RESERVA>=TRUNC(SYSDATE@!-365)				

Tiempo obtenido:

Todas las Filas Recuperadas: 88 en 0,043 segundos

Caso 2: Ahora, se probará con valores mayores, servicios que hayan sido solicitados menos de 20 veces semanales

Resultado:

	NOMBRE_DEL_SERVICIO
163	Mojito
164	Fish and Chips
165	Gin and Tonic
166	Caesar Salad
167	earrings
168	Pad Thai
169	Mushroom Pizza
170	soda
171	tiara
172	chicken
173	Makeup Application
174	Lemon Herb Roaste...
175	yogurt
176	Lobster Bisque
177	waist bag
178	cufflinks
179	chandelier earrings
180	Fish Tacos
181	watch chain
182	White Russian
183	Gimnasio
184	ice cream
185	bangle

Registros retornados: 185

Valores de los parámetros: Servicios que hayan sido solicitados menos de 20 veces semanales

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			10	515
FILTER				
Filter Predicates				
SUM(CANTIDAD_DE_SOLICITUDES)/COUNT(CANTIDAD_DE_SOLICITUDES)<20				
HASH		GROUP BY	10	515
VIEW			47355	515
HASH		GROUP BY	47355	515
TABLE ACCESS	RESERVA_SERVICIO	FULL	48815	223
Filter Predicates				
RS.FECHA_RESERVA>=TRUNC(SYSDATE@!-365)				

Tiempo obtenido:

Todas las Filas Recuperadas: 185 en 0,052 segundos

9. RF9

Para el RF9 Se quiere conocer la información de los clientes que consumieron al menos una vez un determinado servicio del hotel, en un rango de fechas.

Caso 1: Se intentará encontrar los clientes que consumieron mimosa en el mes de julio

Resultado:

SQL | Todas las Filas Recuperadas: 1678 en 0,193 segundos

	NOMBRE	NUMERO_DOCUMENTO	CANTCOMPRAS
1656	Libby Di Dello	3407	2
1657	Rouvin Stathor	3682	1
1658	Nikkie Halworth	853	1
1659	Garfield Clampe	863	1
1660	Gilberte Doget	1043	1
1661	Shep Wordesworth	1255	1
1662	Shelli Linny	1961	3
1663	Willie Scott	3144	1
1664	Eryn Tussaine	4543	1
1665	Munroe Dolphin	1837	1
1666	Xenos Tookill	1971	1
1667	Corney Taysbil	1989	3
1668	Lynette Edscer	3148	1
1669	Marya Strathdee	3810	1
1670	Hersch Haith	4674	1
1671	Randy Edgeworth	4971	1
1672	Liva Rathjen	1554	1
1673	Issy Abbets	2253	2
1674	Alane Powis	2613	1
1675	Ree Lean	3734	1
1676	Sallee Chater	733	1
1677	Selene Hayen	1802	1
1678	Charo Luna	3522	1

Registros retornados: 1678

Valores de los parámetros: clientes que consumieron mimosa en el mes de julio

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1985	259
HASH JOIN			1985	259
Access Predicates				
USUARIOS.NUMERO_DOCUMENTO=ITEM_1				
TABLE ACCESS	USUARIOS	FULL	1985	19
VIEW	SYS.VW_GBF_17		1985	240
HASH		GROUP BY	1985	240
FILTER				
Filter Predicates				
TO_DATE('01/jul/2023')<=TO_DATE('31/julio/2023')				
NESTED LOOPS			5147	239
TABLE ACCESS	RESERVA_SERVICIO	FULL	24	215
Filter Predicates				
AND				
RESERVA_SERVICIO.SERVICIOS_NOMBRE='Mimosa'				
RESERVA_SERVICIO.FECHA_RESERVA>='01/jul/2023'				
RESERVA_SERVICIO.FECHA_RESERVA<='31/julio/2023'				
INDEX	RESERVA_HABITACION_PK	RANGE SCAN	215	1
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				

Tiempo obtenido:

Todas las Filas Recuperadas: 1678 en 0,211 segundos

Caso 2: Ahora, se probará quienes hayan usado el servicio de piscina el día 17 de mayo

Resultado:

	NOMBRE	NUMERO_DOCUMENTO	CANTCOMPRAS
204	Nestor Verrell	1196	1
205	Murry Papworth	1600	1
206	Maryl Calliss	1001	1
207	Christoforo Standall	1838	1
208	Port Todari	1920	1
209	Padgett Duddan	1416	1
210	Hilliard Amsberger	1375	1
211	Shoshana Romera	1057	1
212	Borden Tomei	1926	1
213	Talbot Liebermann	1316	1
214	Kassi Pigram	1532	1
215	Debora Oxborrow	1875	1
216	Loria Twigger	1141	1
217	Lena Cultcheth	1684	1
218	Concettina Perrygo	1061	1
219	Aurel Lugg	1209	1
220	Domini Gregan	1881	1
221	Sharon Dahmel	1756	1
222	Niko Shawley	4664	1
223	Eadith Entwhistle	4107	1
224	Leonelle Tucker	2023	1
225	Sandi Taylor	2470	1
226	Kelwin Hasslocher	2998	1

Registros retornados: 226

Valores de los parámetros: Servicio piscina, el día 17 de mayo

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			153	236
HASH		GROUP BY	153	236
FILTER				
Filter Predicates				
TO_DATE('17/may/2023')>=TO_DATE('17/may/2023')				
HASH JOIN			153	235
Access Predicates				
USUARIOS.NUMERO_DOCUMENTO=RESERVA_HABITACION.USUARIOS_NUMERO_DOCUMENTO				
HASH JOIN			153	216
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
NESTED LOOPS			153	216
STATISTICS COLLECTOR				
TABLE ACCESS	RESERVA_SERVICIO	FULL	1	215
Filter Predicates				
AND				
RESERVA_SERVICIO.SERVICIOS_NOMBRE='Piscina'				
RESERVA_SERVICIO.FECHA_RESERVA='17/may/2023'				
INDEX	RESERVA_HABITACION_PK	RANGE SCAN	215	1
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
INDEX	RESERVA_HABITACION_PK	FAST FULL SCAN	215	1
TABLE ACCESS	USUARIOS	FULL	1985	19

Tiempo obtenido:

Todas las Filas Recuperadas: 226 en 0,047 segundos

10.RF10

Para el RF10 Se quiere conocer la información de los clientes que NO consumieron al menos una vez un determinado servicio del hotel, en un rango de fechas.

Caso 1: Se intentará encontrar los clientes que NO consumieron mimosa en el mes de julio

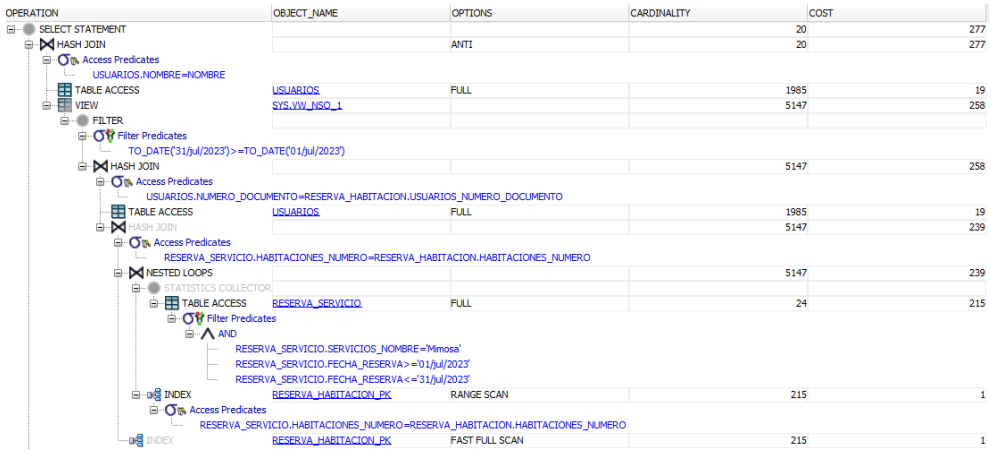
Resultado:

	NOMBRE	NUMERO_DOCUMENTO
285	Cybil Emerson	1128
286	Harvey Eddis	942
287	Robby Goodings	1199
288	Kane Millom	1338
289	Giusto Casford	525
290	Damian Goude	3115
291	Mead Youthed	1097
292	Gradeigh Sevin	296
293	Humphrey Pleaden	1590
294	Ilka Coger	209
295	Scotti Blondel	1066
296	Giralda Mibourne	57
297	Sigfrid Mintoft	182
298	Lelah Antunez	3169
299	Barnard Artz	384
300	Patty Gadesby	195
301	Miner Splevin	2844
302	Anstice Kolodziej	3689
303	Kim McMurthy	538
304	Bethanne Fasson	533
305	Karole Kurtis	336
306	Dee Croce	4142
307	Franny Fish	2839

Registros retornados: 307

Valores de los parámetros: clientes que NO consumieron mimosa en el mes de julio

Plan de consulta:



Tiempo obtenido:

Todas las Filas Recuperadas: 307 en 0,058 segundos

Caso 2: Ahora, se probará quienes NO hayan usado el servicio de piscina el día 17 de mayo

Resultado:

	NOMBRE	NUMERO_DOCUMENTO
1736	Erasmus Neasam	4570
1737	Pierce Upex	2729
1738	Lynnett Crowche	4780
1739	Enoch Rawsthorn	4265
1740	Sheppard Wenden	1214
1741	Nora Greenmon	3564
1742	Calypso Sylett	4204
1743	Louise Stubley	4688
1744	Kristopher Rowell	3614
1745	Boycey Herion	3804
1746	Franzen Ravenhill	2843
1747	Zaria Neiland	3656
1748	Hynda Keam	1552
1749	Tarra Christoforou	3676
1750	Reiko Crossfield	2143
1751	Mora MacPadene	2089
1752	Wildon Pellatt	4271
1753	Darlleen Eakin	2507
1754	Quinton McLoney	2357
1755	Rorke Soldner	1841
1756	Lorry Onions	1666
1757	Thibaut Inglese	2820
1758	Cornell Cashell	1452
1759	Salaidh Spindler	3600

Registros retornados: 1759

Valores de los parámetros: Quienes no hayan usado el Servicio piscina, el día 17 de mayo

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1832	254
HASH JOIN		RIGHT ANTI	1832	254
Access Predicates				
USUARIOS.NOMBRE=NOMBRE				
VIEW	SYS.VW_NSO_1		153	235
FILTER				
Filter Predicates				
TO_DATE('17/may/2023')>=TO_DATE('17/may/2023')				
HASH JOIN			153	235
Access Predicates				
USUARIOS.NUMERO_DOCUMENTO=RESERVA_HABITACION.USUARIOS_NUMERO_DOCUMENTO				
HASH JOIN			153	216
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
NESTED LOOPS			153	216
STATISTICS COLLECTOR				
TABLE ACCESS	RESERVA_SERVICIO	FULL	1	215
Filter Predicates				
AND				
RESERVA_SERVICIO.SERVICIOS_NOMBRE='Piscina'				
RESERVA_SERVICIO.FECHA_RESERVA='17/may/2023'				
INDEX	RESERVA_HABITACION_PK	RANGE SCAN	215	1
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
INDEX	RESERVA_HABITACION_PK	FAST FULL SCAN	215	1
TABLE ACCESS	USUARIOS	FULL	1985	19
TABLE ACCESS	USUARIOS	FULL	1985	19

Tiempo obtenido:

Todas las Filas Recuperadas: 1759 en 0,179 segundos

11.RF11

Para el RF11 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.

Caso 1: Todas las semanas del año

Resultado:

	NUMERO_SEMANA	SERVICIO_MAS_CONSUMIDO	SERVICIO_MENOS_CONSUMIDO
23	23	yogurt	Aromatherapy
24	24	whiskey	Acupuncture
25	25	whiskey	Acupuncture
26	26	yogurt	Aromatherapy
27	27	yogurt	Beef Burrito
28	28	yogurt	Aromatherapy
29	29	yogurt	Acupuncture
30	30	wine	Acupuncture
31	31	wine	BBQ Ribs
32	32	whiskey	Beef Bulgogi
33	33	yogurt	Beef Bulgogi
34	34	yogurt	Acupuncture
35	35	yogurt	Acupuncture
36	36	wine	Aromatherapy
37	37	wine	BBQ Ribs
38	38	yogurt	Acupuncture
39	39	waistcoat	Acupuncture
40	40	yogurt	Aromatherapy
41	41	wine	Beef Bulgogi
42	42	yogurt	Acupuncture
43	43	wine	Acupuncture
44	44	yogurt	Beef Burrito

Registros retornados: 44

Valores de los parámetros: Todas las semanas del año

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			53	221
SORT		GROUP BY	53	221
HASH JOIN			7143	220
Access Predicates				
C.FECHA=RS.FECHA_RESERVA				
TABLE ACCESS	CALENDARIO	FULL	53	4
TABLE ACCESS	RESERVA_SERVICIO	FULL	245965	215

Tiempo obtenido:

Todas las Filas Recuperadas: 44 en 0,047 segundos

Caso 2: Ahora, solo las primeras 20 semanas del año

Resultado:

	NUMERO_SEMANA	SERVICIO_MAS_CONSUMIDO	SERVICIO_MENOS_CONSUMIDO
1	1	yogurt	Acupuncture
2	2	yogurt	Acupuncture
3	3	yogurt	Acupuncture
4	4	yogurt	Acupuncture
5	5	water	Acupuncture
6	6	wine	Acupuncture
7	7	wine	Acupuncture
8	8	yogurt	Beef Bulgogi
9	9	yogurt	Acupuncture
10	10	yogurt	Beef Burrito
11	11	yogurt	Beef Bulgogi
12	12	whiskey	Beef Bulgogi
13	13	yogurt	Acupuncture
14	14	yogurt	Acupuncture
15	15	yogurt	Acupuncture
16	16	water	BBQ Ribs
17	17	whiskey	Aromatherapy
18	18	water	BBQ Ribs
19	19	yogurt	Beef Bulgogi
20	20	wine	BBQ Ribs

Registros retornados: 20

Valores de los parámetros: Las primeras 20 semanas del año

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			20	221
VIEW	SYS.null		20	221
Filter Predicates				
from$_subquery$_006.rowlimit_$$_rownumber<=20				
WINDOW		NOSORT STOPKEY	53	221
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY C.NUMERO_SEMANA)<=20				
SORT		GROUP BY	53	221
HASH JOIN			7143	220
Access Predicates				
C.FECHA=RS.FECHA_RESERVA				
TABLE ACCESS	CALENDARIO	FULL	53	4
TABLE ACCESS	RESERVA_SERVICIO	FULL	245965	215

Tiempo obtenido:

Todas las Filas Recuperadas: 20 en 0,033 segundos

12.RF12

Para el RF12 muestra los clientes excelentes, Los clientes excelentes 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.

Caso 1: Clientes excelentes sin más

Resultado:

	NUMERO_DOCUMENTO	NOMBRE	TIPO_EXCELENTE
1968	4204	Calypso Sylett	Excelente (Servicio Costoso)
1969	4688	Louise Stubley	Excelente (Servicio Costoso)
1970	3614	Kristopher Rowell	Excelente (Servicio Costoso)
1971	3804	Boycey Herion	Excelente (Servicio Costoso)
1972	2843	Franzen Ravenhill	Excelente (Servicio Costoso)
1973	3656	Zaria Neiland	Excelente (Servicio Costoso)
1974	1552	Hynda Keam	Excelente (Servicio Costoso)
1975	3676	Tarra Christoforou	Excelente (Servicio Costoso)
1976	2143	Reiko Crossfield	Excelente (Servicio Costoso)
1977	2089	Mora MacPadene	Excelente (Servicio Costoso)
1978	4271	Wildon Pellatt	Excelente (Servicio Costoso)
1979	2507	Darlleen Eakin	Excelente (Servicio Costoso)
1980	2357	Quinton McLoney	Excelente (Servicio Costoso)
1981	1841	Rorke Soldner	Excelente (Servicio Costoso)
1982	1666	Lorry Onions	Excelente (Servicio Costoso)
1983	2820	Thibaut Inglese	Excelente (Servicio Costoso)
1984	1452	Cornell Cashell	Excelente (Servicio Costoso)
1985	3600	Salaidh Spindler	Excelente (Servicio Costoso)

Registros retornados: 1985

Valores de los parámetros: Clientes excelentes sin más

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1985	3006
HASH JOIN		RIGHT OUTER	1985	3006
Access Predicates				
USUARIOS.NUMERO_DOCUMENTO=S.NUMERO_DOCUMENTO(+)				
VIEW			1985	1781
HASH		UNIQUE	1985	1781
HASH JOIN			23125489	919
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
INDEX	RESERVA_HABITACION_PK	FAST FULL SCAN	215095	187
HASH JOIN			107513	219
Access Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
TABLE ACCESS	SERVICIOS	FULL	81	4
Filter Predicates				
SERVICIOS.COSTO>300000				
TABLE ACCESS	RESERVA_SERVICIO	FULL	245965	215
HASH JOIN		RIGHT OUTER	1985	1225
Access Predicates				
USUARIOS.NUMERO_DOCUMENTO=E.NUMERO_DOCUMENTO(+)				
VIEW			20	1206
FILTER				
Filter Predicates				
COUNT($vm_col_2)=4				
HASH		GROUP BY	20	1206
VIEW	SYS.VM_NWVW_1		215095	1199
HASH		GROUP BY	215095	1199
TABLE ACCESS	RESERVA_HABITACION	FULL	215095	215
TABLE ACCESS	USUARIOS	FULL	1985	19

Tiempo obtenido:

Todas las Filas Recuperadas: 1985 en 1,91 segundos

Caso 2: Ahora, solo Los primeros 100 clientes excelentes ordenados por id

Resultado:

	NUMERO_DOCUMENTO	NOMBRE	TIPO_EXCELENTE
81	104	Tamarah Dmitrovic	Excelente (Servicio Costoso)
82	105	Pryce Chaffyn	Excelente (Servicio Costoso)
83	106	Albertina Trevit...	Excelente (Servicio Costoso)
84	107	Rochelle Brother...	Excelente (Servicio Costoso)
85	108	Neile Overington	Excelente (Servicio Costoso)
86	109	Christen Wheadon	Excelente (Servicio Costoso)
87	110	Fancie Kusick	Excelente (Servicio Costoso)
88	111	Corrianne Wareing	Excelente (Servicio Costoso)
89	112	Beverlie Romayne	Excelente (Servicio Costoso)
90	114	Wendel Edlin	Excelente (Servicio Costoso)
91	115	Rorke Standering	Excelente (Servicio Costoso)
92	116	Fannie Millsom	Excelente (Servicio Costoso)
93	117	Kissiah McClurg	Excelente (Servicio Costoso)
94	118	Dasi McIlriach	Excelente (Servicio Costoso)
95	119	Loreen Crayk	Excelente (Servicio Costoso)
96	121	Tymon Ridolfo	Excelente (Servicio Costoso)
97	122	Doloritas Commucci	Excelente (Servicio Costoso)
98	123	Inglebert Sherlock	Excelente (Servicio Costoso)
99	124	Jere Tolworth	Excelente (Servicio Costoso)
100	126	Lari Geockle	Excelente (Servicio Costoso)

Registros retornados: 100

Valores de los parámetros: Los primeros 100 clientes excelentes ordenados por id

Plan de consulta:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			100	3007
VIEW	SYS.null		100	3007
Filter Predicates				
from$_subquery$_015.rowlimit_$$_rownumber<=100				
WINDOW		SORT PUSHED RANK	1985	3007
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY USUARIOS.NUMERO_DOCUMENTO)<=100				
HASH JOIN		RIGHT OUTER	1985	3006
Access Predicates				
USUARIOS.NUMERO_DOCUMENTO=S.NUMERO_DOCUMENTO(+)				
VIEW			1985	1781
HASH		UNIQUE	1985	1781
HASH JOIN			23125489	919
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
INDEX	RESERVA_HABITACION_PK	FAST FULL SCAN	215095	187
HASH JOIN			107513	219
Access Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
TABLE ACCESS	SERVICIOS	FULL	81	4
Filter Predicates				
SERVICIOS.COSTO>300000				
TABLE ACCESS	RESERVA_SERVICIO	FULL	245965	215
HASH JOIN		RIGHT OUTER	1985	1225
Access Predicates				
USUARIOS.NUMERO_DOCUMENTO=E.NUMERO_DOCUMENTO(+)				
VIEW			100	1206
FILTER				
Filter Predicates				
COUNT($vm_col_2)>=4				
HASH		GROUP BY	100	1206
VIEW	SYS.VM_NWVW_1		215095	1199
HASH		GROUP BY	215095	1199
TABLE ACCE	RESERVA_HABITACION	FULL	215095	215
TABLE ACCESS	USUARIOS	FULL	1985	19

Tiempo obtenido:

Todas las Filas Recuperadas: 100 en 1,753 segundos

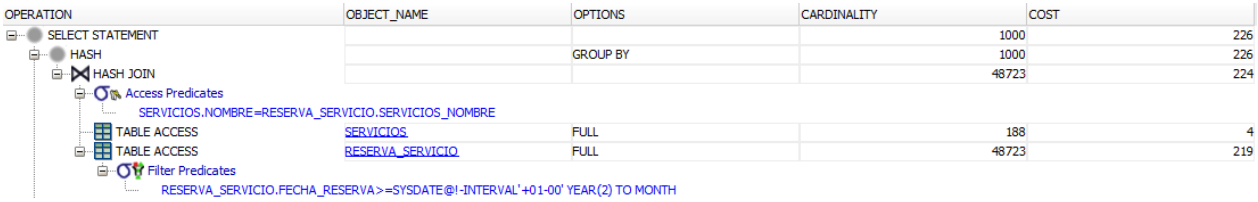
- **Análisis de eficiencia**

1. RF1:

Para el RF1 se propone el siguiente plan de ejecución de la consulta para el caso 1:

1. Selecciona el número de la habitación y la suma de los costos de esta
2. Realizar un JOIN entre las tablas servicios y reserva_servicios como punto de comparación los nombres.
3. Acceder a las tablas de servicios y reserva_servicios, y traer solo los valores del último año

Oracle:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1000	226
HASH		GROUP BY	1000	226
HASH JOIN			48723	224
Access Predicates				
SERVICIOS.NOMBRE=RESERVA_SERVICIO.SERVICIOS_NOMBRE				
TABLE ACCESS	SERVICIOS	FULL	188	4
TABLE ACCESS	RESERVA_SERVICIO	FULL	48723	219
Filter Predicates				
RESERVA_SERVICIO.FECHA_RESERVA>=SYSDATE@!-INTERVAL'+01-00' YEAR(2) TO MONTH				

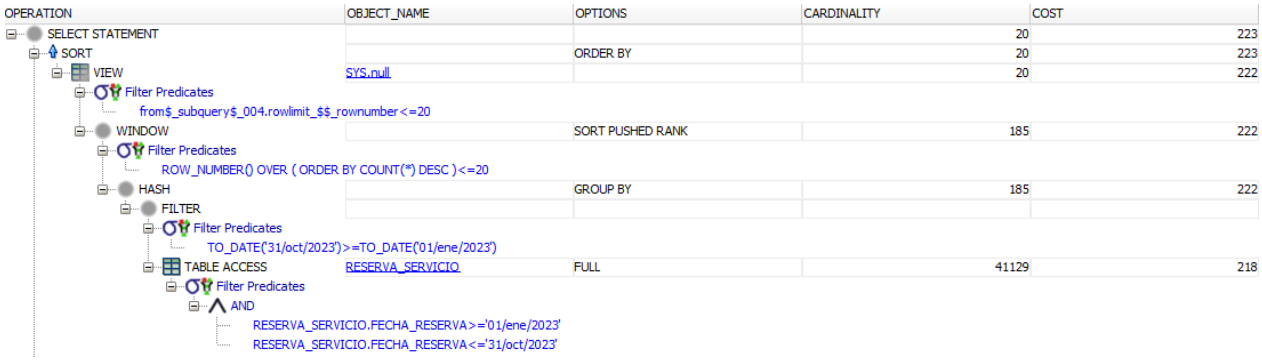
Los planes propuestos son bastante parecidos, ya que se sigue una dinámica de JOIN y de acceso a tablas bastante parecida, esto nos lleva a pensar que puede que sea una manera bastante eficiente de realizar la consulta

2. RF2:

Para el RF2 se propone el siguiente plan de ejecución de la consulta para el caso 2:

1. Selecciona el nombre del servicio y la cantidad de veces que fue usado
2. Filtrar por la fecha de la reserva del servicio
3. Filtrar por los primeros 20 servicios que más fueron usados

Oracle:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			20	223
SORT		ORDER BY	20	223
VIEW	SYS.null		20	222
Filter Predicates				
from$_subquery$_004.rowlimit_$$_rownumber<=20				
WINDOW		SORT PUSHED RANK	185	222
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC)<=20				
HASH		GROUP BY	185	222
FILTER				
Filter Predicates				
TO_DATE('31/oct/2023')>=TO_DATE('01/ene/2023')				
TABLE ACCESS	RESERVA_SERVICIO	FULL	41129	218
Filter Predicates				
AND				
RESERVA_SERVICIO.FECHA_RESERVA>='01/ene/2023'				
RESERVA_SERVICIO.FECHA_RESERVA<='31/oct/2023'				

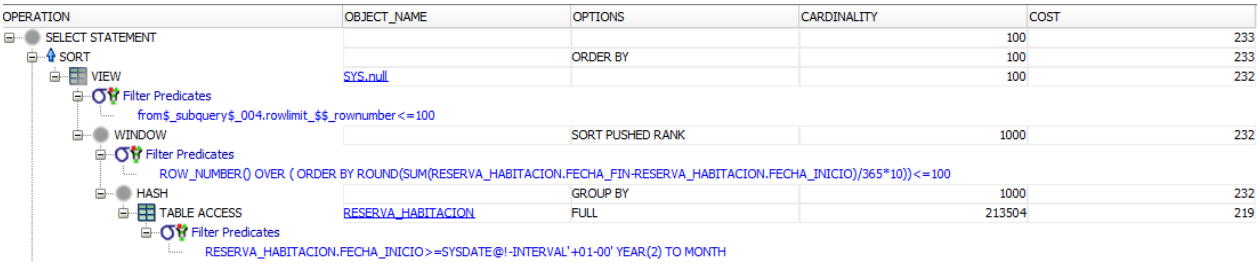
Los planes propuestos son bastante parecidos, solo que Oracle primero filtra por los primeros 20 servicios, y al final si realiza el filtrado de las fechas que están pidiendo

3. **RF3:**

   Para el RF3 se propone el siguiente plan de ejecución de la consulta para el caso 2:

   1. Seleccionar el número de habitación
   2. Realizar las operaciones para hallar el índice de ocupación de las habitaciones
   3. Ordenar los resultados de menor a mayor en índice de ocupación
   4. Filtrar los primeros 100 resultados

Oracle:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			100	233
SORT		ORDER BY	100	233
VIEW	SYS.null		100	232
Filter Predicates				
from$_subquery$_004.rowlimit_$$_rownumber<=100				
WINDOW		SORT PUSHED RANK	1000	232
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY ROUND(SUM(RESERVA_HABITACION.FECHA_FIN-RESERVA_HABITACION.FECHA_INICIO)/365*10))<=100				
HASH		GROUP BY	1000	232
TABLE ACCESS	RESERVA_HABITACION	FULL	213504	219
Filter Predicates				
RESERVA_HABITACION.FECHA_INICIO>=SYSDATE@!-INTERVAL'+01-00' YEAR(2) TO MONTH				

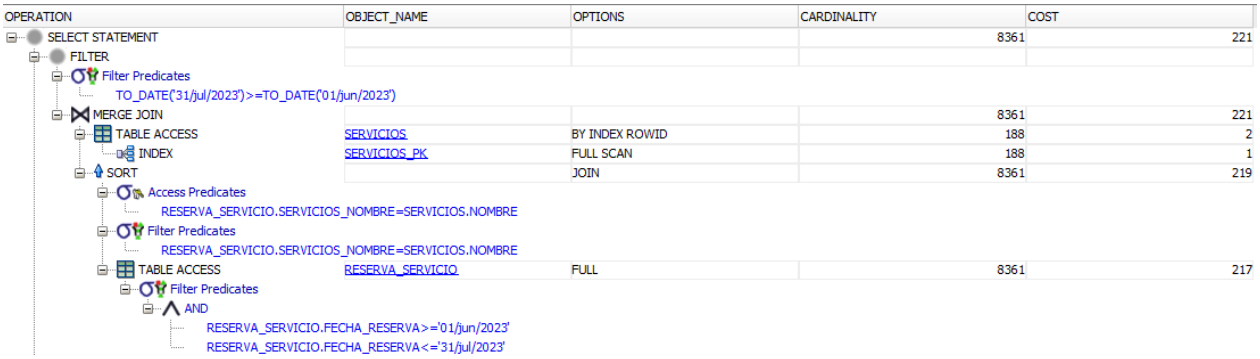
El plan de Oracle propone realizar las operaciones de order by y las restas, divisiones y multiplicaciones en un solo paso, lo que haría que la consulta fuera mucho más eficiente. También accede a la tabla de reserva para saber cuales de las habitaciones fueron ocupadas durante el último año

4. RF4:

Para el RF4 se propone el siguiente plan de ejecución de la consulta para el caso 2:

1. Seleccionar todas las columnas del servicio y de la reserva
2. Realizar un HASH JOIN entre estas dos tablas
3. Filtrar entre las fechas que están pidiendo

Oracle:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			8361	221
FILTER				
Filter Predicates				
TO_DATE('31/jul/2023')>=TO_DATE('01/jun/2023')				
MERGE JOIN			8361	221
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	188	2
INDEX	SERVICIOS_PK	FULL SCAN	188	1
SORT		JOIN	8361	219
Access Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
Filter Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
TABLE ACCESS	RESERVA_SERVICIO	FULL	8361	217
Filter Predicates				
AND				
RESERVA_SERVICIO.FECHA_RESERVA>='01/jun/2023'				
RESERVA_SERVICIO.FECHA_RESERVA<='31/jul/2023'				

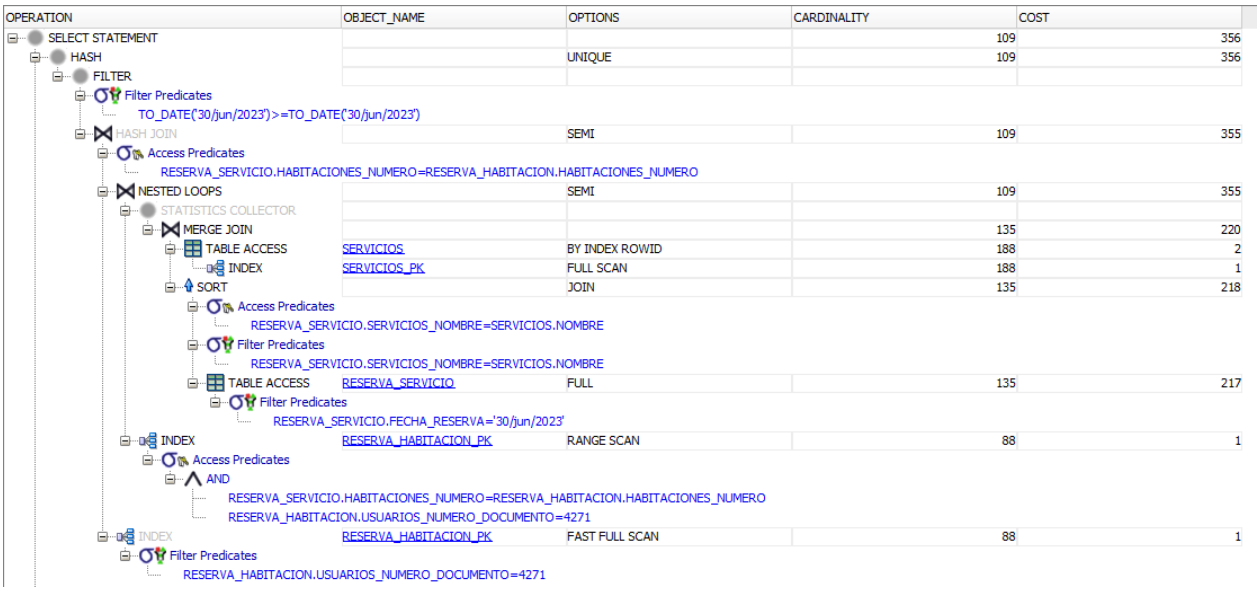
El plan de Oracle propone realizar el JOIN mediante un MERGE JOIN, y filtra los predicados de las fechas al principio, esto puede suceder ya que si filtra primero las fechas no tendría que acceder a todos los servicios, así realizando una consulta mucho más eficiente.

5. RF5:

Para el RF5 se propone el siguiente plan de ejecución de la consulta para el caso 2:

1. Seleccionar el nombre de los servicios y el costo de estos
2. Filtrar las fechas para que solo queden los meses de junio y julio
3. Realizar un HASH JOIN entre la tabla usuario y reserva habitación
4. Realizar un HASH JOIN entre la tabla reserva habitación y habitación
5. Realizar un HASH JOIN entre la tabla habitación y reserva servicio
6. Realizar un HASH JOIN entre la tabla reserva servicio y servicio
7. Encontrar el usuario con id 4271

Oracle:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			109	356
HASH		UNIQUE	109	356
FILTER				
Filter Predicates				
TO_DATE('30/jun/2023')>=TO_DATE('30/jun/2023')				
HASH JOIN		SEMI	109	355
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
NESTED LOOPS		SEMI	109	355
STATISTICS COLLECTOR				
MERGE JOIN			135	220
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	188	2
INDEX	SERVICIOS_PK	FULL SCAN	188	1
SORT		JOIN	135	218
Access Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
Filter Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
TABLE ACCESS	RESERVA_SERVICIO	FULL	135	217
Filter Predicates				
RESERVA_SERVICIO.FECHA_RESERVA='30/jun/2023'				
INDEX	RESERVA_HABITACION_PK	RANGE SCAN	88	1
Access Predicates				
AND				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
RESERVA_HABITACION.USUARIOS_NUMERO_DOCUMENTO=4271				
INDEX	RESERVA_HABITACION_PK	FAST FULL SCAN	88	1
Filter Predicates				
RESERVA_HABITACION.USUARIOS_NUMERO_DOCUMENTO=4271				

El plan de Oracle propone realizar el JOIN mediante un MERGE JOIN, y filtra los predicados de las fechas al principio, esto puede suceder ya que si filtra primero las fechas no tendría que acceder a todos los servicios, así realizando una consulta mucho más eficiente. También realiza un sort en

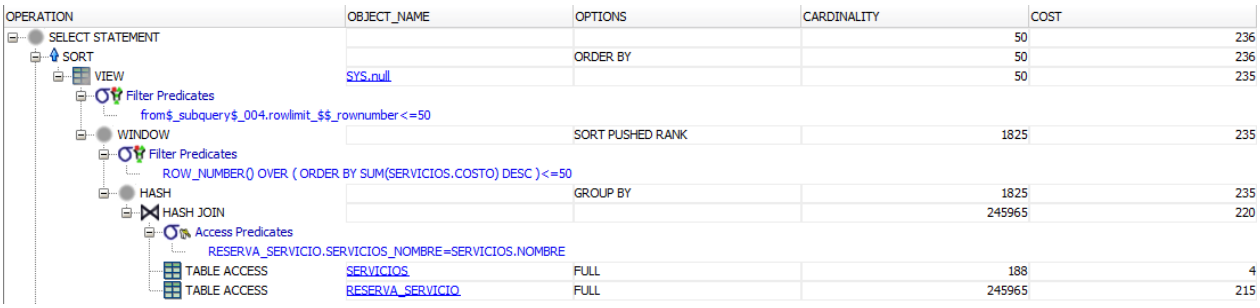
la tabla de reserva servicios para poder hacer el JOIN con la tabla servicios de manera más eficiente. Al final es que encuentra al usuario con documento 4271

6. RF6:

Para el RF6 se propone el siguiente plan de ejecución de la consulta para el caso 3:

1. Seleccionar la fecha de la reserva del servicio
2. Realizar la suma de todos los costos que se cobraron en esa fecha
3. Realizar un HASH JOIN entre la tabla reserva servicio y servicio
4. Filtrar los primeros 50 registros ordenados de mayor a menor

Oracle:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			50	236
SORT		ORDER BY	50	236
VIEW	SYS.null		50	235
Filter Predicates				
from$_subquery$_004.rowlimit_$$_rownumber<=50				
WINDOW		SORT PUSHED RANK	1825	235
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY SUM(SERVICIOS.COSTO) DESC)<=50				
HASH		GROUP BY	1825	235
HASH JOIN			245965	220
Access Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
TABLE ACCESS	SERVICIOS	FULL	188	4
TABLE ACCESS	RESERVA_SERVICIO	FULL	245965	215

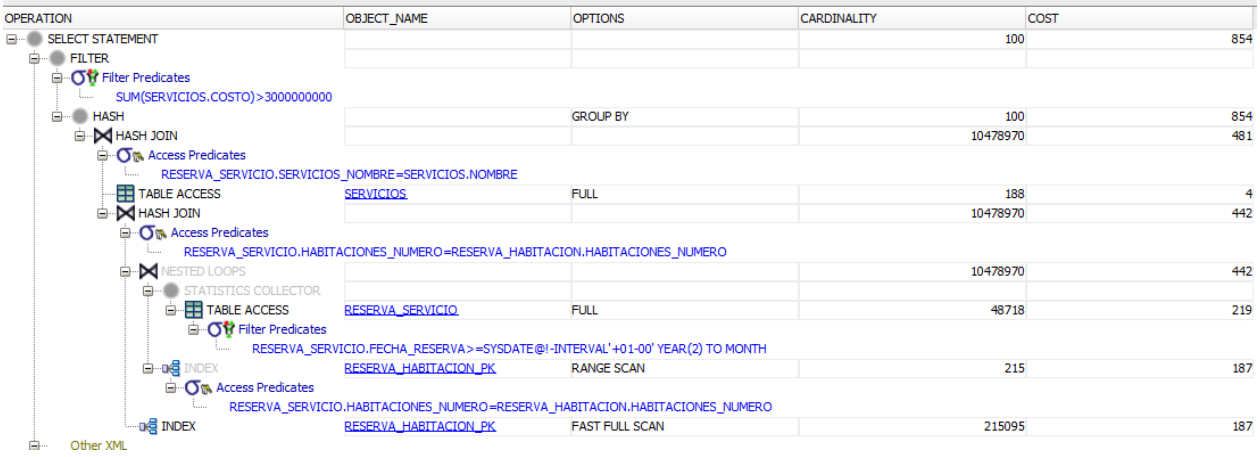
El plan de Oracle propone realizar el JOIN mediante un HASH JOIN, al igual que lo propuesto por el grupo y filtra los primeros valores al principio. También se observa que la operación de filtrado y de la suma de los costos lo realiza todo en un solo paso, para así realizar la consulta de forma más eficiente

7. RF7:

Para el RF7 se propone el siguiente plan de ejecución de la consulta para el caso 2:

1. Seleccionar el número de documento del usuario
2. Realizar un HASH JOIN entre la tabla usuario y reserva habitación
3. Realizar un HASH JOIN entre la tabla reserva habitación y habitación
4. Realizar un HASH JOIN entre la tabla habitación y reserva servicio
5. Realizar un HASH JOIN entre la tabla reserva servicio y servicio
6. Realizar la suma de todos los servicios que el usuario realizó
7. Filtrar solo los servicios que se realizaron en el último año
8. Filtrar que la suma de los costos sea mayor a 3000 millones

Oracle:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			100	854
FILTER				
Filter Predicates				
SUM(SERVICIOS.COSTO)>3000000000				
HASH		GROUP BY	100	854
HASH JOIN			10478970	481
Access Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
TABLE ACCESS	SERVICIOS	FULL	188	4
HASH JOIN			10478970	442
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
NESTED LOOPS			10478970	442
STATISTICS COLLECTOR				
TABLE ACCESS	RESERVA_SERVICIO	FULL	48718	219
Filter Predicates				
RESERVA_SERVICIO.FECHA_RESERVA>=SYSDATE@!-INTERVAL'+01-00' YEAR(2) TO MONTH				
INDEX	RESERVA_HABITACION_PK	RANGE SCAN	215	187
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
INDEX	RESERVA_HABITACION_PK	FAST FULL SCAN	215095	187
Other XML				

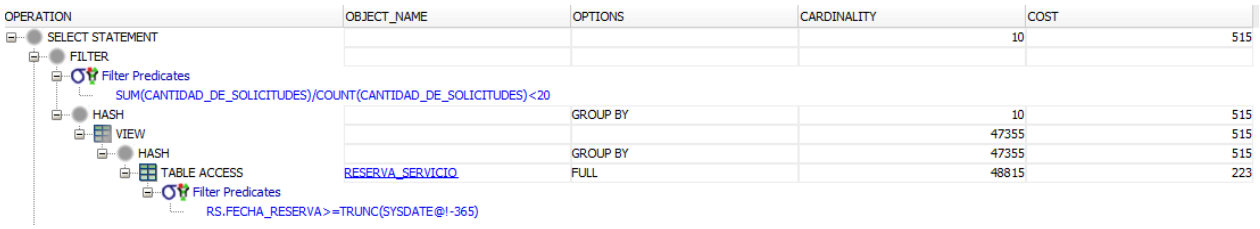
El plan de Oracle propone realizar los JOIN mediante HASH JOIN, al igual que lo propuesto por el grupo y filtra los valores al principio. También se observa que la operación de filtrado y de la suma de los costos lo realiza todo en un solo paso, para así realizar la consulta de forma más eficiente. Se ve que usa el índice de reserva_habitacion_pk para realizar un range scan, y así realizar la consulta de manera más eficiente.

8. RF8:

Para el RF8 se propone el siguiente plan de ejecución de la consulta para el caso 2:

1. Seleccionar el nombre del servicio
2. Hacer HASH JOIN entre servicios y reserva servicios
3. Contar el número de solicitudes que hayan tenido los servicios
4. Contar semanalmente los servicios
5. Filtrar solo los servicios desde hace un año

Oracle:



El plan de Oracle propone realizar los JOIN mediante HASH JOIN, al igual que lo propuesto por el grupo y filtra los valores al principio. También se observa que la operación de filtrado y de la suma de los servicios lo realiza todo en un solo paso, para así realizar la consulta de forma más eficiente. Al final se filtra por las fechas que pide

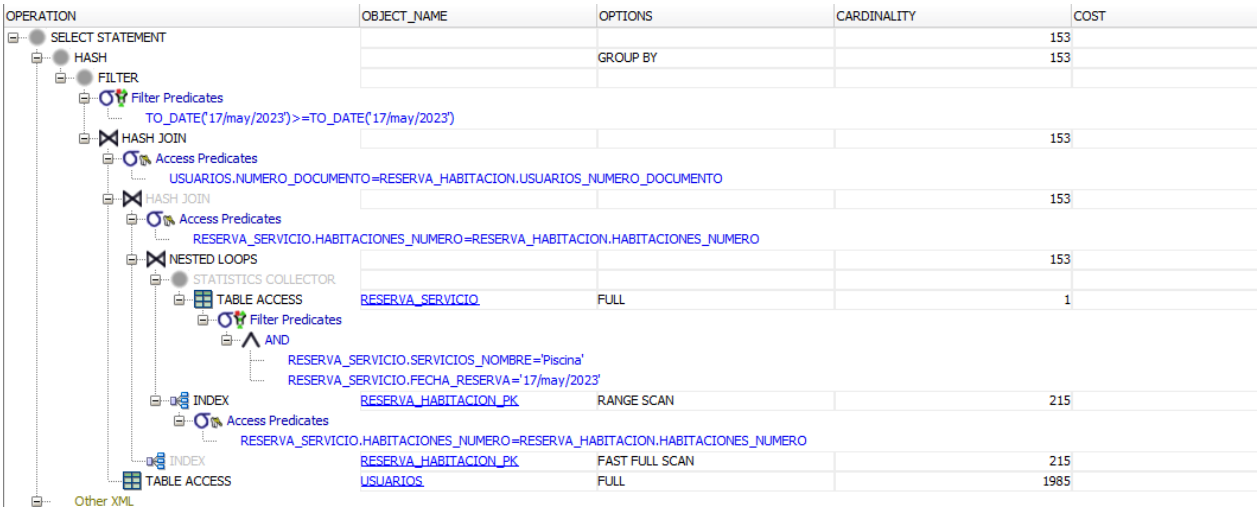
9. RF9:

Para el RF9 se propone el siguiente plan de ejecución de la consulta para el caso 2:

1. Seleccionar el nombre del usuario y su id
2. Realizar un HASH JOIN entre la tabla usuario y reserva habitación
3. Realizar un HASH JOIN entre la tabla reserva habitación y habitación
4. Realizar un HASH JOIN entre la tabla habitación y reserva servicio
5. Realizar un HASH JOIN entre la tabla reserva servicio y servicio

6. Filtrar por la fecha que se pide
7. Filtrar por el nombre del servicio = “Piscina”

Oracle:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			153	
HASH		GROUP BY	153	
FILTER				
Filter Predicates				
TO_DATE('17/may/2023')>=TO_DATE('17/may/2023')				
HASH JOIN			153	
Access Predicates				
USUARIOS.NUMERO_DOCUMENTO=RESERVA_HABITACION.USUARIOS_NUMERO_DOCUMENTO				
HASH JOIN			153	
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
NESTED LOOPS			153	
STATISTICS COLLECTOR				
TABLE ACCESS	RESERVA_SERVICIO	FULL	1	
Filter Predicates				
AND				
RESERVA_SERVICIO.SERVICIOS_NOMBRE='Piscina'				
RESERVA_SERVICIO.FECHA_RESERVA='17/may/2023'				
INDEX	RESERVA_HABITACION_PK	RANGE SCAN	215	
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
INDEX	RESERVA_HABITACION_PK	FAST FULL SCAN	215	
TABLE ACCESS	USUARIOS	FULL	1985	
Other XML				

El plan de Oracle propone realizar los JOIN mediante HASH JOIN, al igual que lo propuesto por el grupo y filtra los valores de fecha al principio. También se observa que se accede a la tabla reserva_servicio mediante un nested loop, y que se hace un range scan gracias al índice de esta tabla.

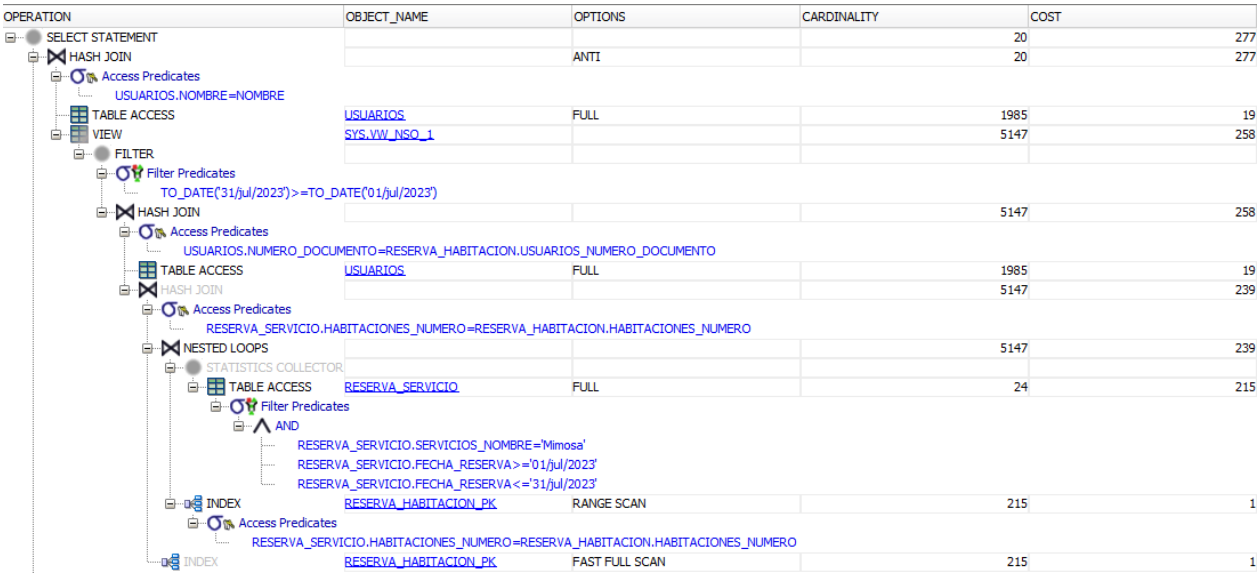
10.RF10:

Para el RF10 se propone el siguiente plan de ejecución de la consulta para el caso 1:

1. Seleccionar el nombre del usuario y su id
2. Realizar un HASH JOIN entre la tabla usuario y reserva habitación
3. Realizar un HASH JOIN entre la tabla reserva habitación y habitación

4. Realizar un HASH JOIN entre la tabla habitación y reserva servicio
5. Realizar un HASH JOIN entre la tabla reserva servicio y servicio
6. Filtrar por la fecha que se pide, mes de julio
7. Filtrar por el nombre del servicio = “Mimosa”

Oracle:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			20	277
HASH JOIN		ANTI	20	277
Access Predicates				
USUARIOS.NOMBRE=NOMBRE				
TABLE ACCESS	USUARIOS	FULL	1985	19
VIEW	SYS.VW_NSO_1		5147	258
FILTER				
Filter Predicates				
TO_DATE('31/jul/2023')>=TO_DATE('01/jul/2023')				
HASH JOIN			5147	258
Access Predicates				
USUARIOS.NUMERO_DOCUMENTO=RESERVA_HABITACION.USUARIOS_NUMERO_DOCUMENTO				
TABLE ACCESS	USUARIOS	FULL	1985	19
HASH JOIN			5147	239
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
NESTED LOOPS			5147	239
STATISTICS COLLECTOR				
TABLE ACCESS	RESERVA_SERVICIO	FULL	24	215
Filter Predicates				
AND				
RESERVA_SERVICIO.SERVICIOS_NOMBRE='Mimosa'				
RESERVA_SERVICIO.FECHA_RESERVA>='01/jul/2023'				
RESERVA_SERVICIO.FECHA_RESERVA<='31/jul/2023'				
INDEX	RESERVA_HABITACION_PK	RANGE SCAN	215	1
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
INDEX	RESERVA_HABITACION_PK	FAST FULL SCAN	215	1

El plan de Oracle propone realizar los JOIN mediante HASH JOIN, al igual que lo propuesto por el grupo y filtra los valores de fecha al principio. También se observa que se accede a la tabla reserva_servicio mediante un nested loop, y que se hace un range scan gracias al índice de esta tabla. Se evidencia que al principio compara los nombres de los usuarios ya que se está haciendo una subconsulta, y quiere es mostrar los valores que no hacen parte de esta subconsulta

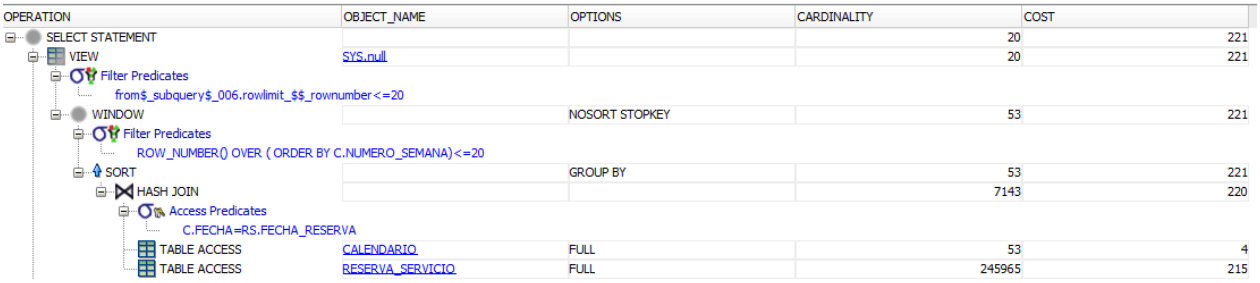
11.RF11:

Para el RF11 se propone el siguiente plan de ejecución de la consulta para el caso 2, (cabe aclarar que para este requerimiento se creó la tabla calendario):

1. Seleccionar el numero de la semana y el nombre del servicio
2. Realizar un HASH JOIN entre la tabla reserva servicio y la tabla calendario

3. Realizar una operación para contar los servicios que fueron consumidos durante el año
4. Filtrar los primeros 20 resultados

Oracle:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			20	221
VIEW	SYS.null		20	221
Filter Predicates				
from$_subquery$_006.rowlimit_$$_rownumber<=20				
WINDOW		NOSORT STOPKEY	53	221
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY C.NUMERO_SEMANA)<=20				
SORT		GROUP BY	53	221
HASH JOIN			7143	220
Access Predicates				
C.FECHA=RS.FECHA_RESERVA				
TABLE ACCESS	CALENDARIO	FULL	53	4
TABLE ACCESS	RESERVA_SERVICIO	FULL	245965	215

El plan de Oracle propone realizar los JOIN mediante HASH JOIN, al igual que lo propuesto por el grupo y filtra los valores de los registros al principio. También se observa que se realiza una operación de group by, ya que se están agrupando las semanas del año.

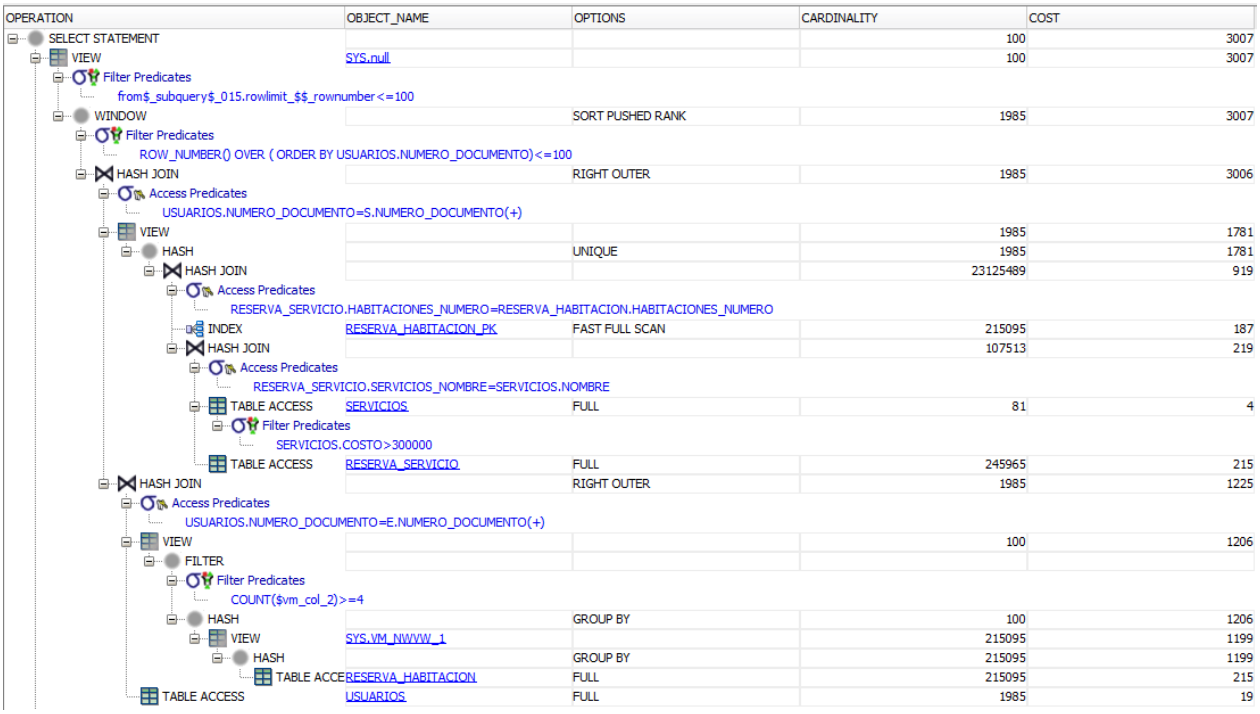
12.RF12:

Para el RF12 se propone el siguiente plan de ejecución de la consulta para el caso 2):

1. Seleccionar el número de documento y nombre del usuario
2. Realizar un HASH JOIN entre la tabla usuario y reserva habitación
3. Realizar un HASH JOIN entre la tabla reserva habitación y habitación
4. Realizar un HASH JOIN entre la tabla habitación y reserva servicio
5. Realizar un HASH JOIN entre la tabla reserva servicio y servicio
6. Subconsulta para verificar estancia
7. Subconsulta para verificar costoso
8. Filtrar por el último año
9. Ordenar los usuarios por ID

10.Filtrar los primeros 100 resultados

Oracle:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			100	3007
VIEW	SYS.null		100	3007
Filter Predicates				
from$_subquery$_015.rowlimit_$$_rownumber<=100				
WINDOW		SORT PUSHED RANK	1985	3007
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY USUARIOS.NUMERO_DOCUMENTO)<=100				
HASH JOIN		RIGHT OUTER	1985	3006
Access Predicates				
USUARIOS.NUMERO_DOCUMENTO=S.NUMERO_DOCUMENTO(+)				
VIEW			1985	1781
HASH		UNIQUE	1985	1781
HASH JOIN			23125489	919
Access Predicates				
RESERVA_SERVICIO.HABITACIONES_NUMERO=RESERVA_HABITACION.HABITACIONES_NUMERO				
INDEX	RESERVA_HABITACION_PK	FAST FULL SCAN	215095	187
HASH JOIN			107513	219
Access Predicates				
RESERVA_SERVICIO.SERVICIOS_NOMBRE=SERVICIOS.NOMBRE				
TABLE ACCESS	SERVICIOS	FULL	81	4
Filter Predicates				
SERVICIOS.COSTO>300000				
TABLE ACCESS	RESERVA_SERVICIO	FULL	245965	215
HASH JOIN		RIGHT OUTER	1985	1225
Access Predicates				
USUARIOS.NUMERO_DOCUMENTO=E.NUMERO_DOCUMENTO(+)				
VIEW			100	1206
FILTER				
Filter Predicates				
COUNT($vm_col_2)>=4				
HASH		GROUP BY	100	1206
VIEW	SYS.VM_NWVW_1		215095	1199
HASH		GROUP BY	215095	1199
TABLE ACCE	RESERVA_HABITACION	FULL	215095	215
TABLE ACCESS	USUARIOS	FULL	1985	19

El plan de Oracle propone realizar los JOIN mediante HASH JOIN, al igual que lo propuesto por el grupo y filtra los valores de los registros al principio. También se observa que se al ser realizado en varias subconsultas, se hacen HASH JOINS con los resultados de esas subconsultas al final del plan.

- **Documentación proceso carga de datos**

Para la creación de los datos de prueba durante esta iteración del proyecto, primero se crearon 1000 habitaciones, enumeradas de 1 a 1000. Además, se crearon 2000 usuarios, con números de identificación entre 1 y 5000, tomados aleatoriamente. Conociendo los PKs de las tablas usuarios y habitaciones, se escribió un pequeño código en python para generar 250.000 registros en la tabla reserva_habitacion. El código creado tiene en cuenta que la fecha de inicio de la reserva va antes de la fecha de fin de la reserva. Además, se tuvo en cuenta que la reserva es corta, de no más de 15 días en promedio. Sin embargo, no se tuvo en cuenta que, para reservar una habitación durante ciertas fechas, no puede estar reservada por otro usuario. Los datos en las columnas de la tabla usuarios, como el nombre del usuario, correo, etc, fueron creados con ayuda de la herramienta ofrecida por https://www.mockaroo.com/ para generar datos que tienen sentido, con ayuda de una inteligencia artificial.

A continuación, se crearon 189 servicios con nombres que tienen sentido dados los servicios que se dice ofrece el hotel en el caso de estudio. Los precios de los servicios fueron generados aleatoriamente, en un rango de entre los 10.000 y 500.000 pesos. Después, y una vez más, conociendo los PKs Habitación.numero y servicios.nombre, se creó un pequeño código para generar 250.000 datos, para poder poblar la tabla reserva_servicio. A continuación, se presenta el código usado para la generación de datos:

```
import random
import datetime

# Listas de posibles valores para las llaves foráneas
habitaciones_numero = valores_habitaciones
servicios_nombre = valores_servicios

# Número de registros que deseas generar
N = 25

# Genera instrucciones INSERT de forma aleatoria para poblar la tabla
reserva_servicio
inserts = []

for _ in range(N):
    # Selecciona aleatoriamente valores de las llaves foráneas
    habitacion = random.choice(habitaciones_numero)
    servicio = random.choice(servicios_nombre)

    # Genera una fecha de reserva aleatoria
    fecha_reserva = datetime.date.today() -
datetime.timedelta(days=random.randint(1, 365*5))
```

```
    # Genera un código aleatorio
    codigo = random.randint(1, 10000)

    # Crea la instrucción INSERT y agrégala a la lista
    insert = f"INSERT INTO reserva_servicio (habitaciones_numero,
servicios_nombre, fecha_reserva, codigo) " \
             f"VALUES ({habitacion}, '{servicio}',
TO_DATE('{fecha_reserva}', 'YYYY-MM-DD'), {codigo});"
    inserts.append(insert)
    with open('datos_reserva_servicio.sql', 'a') as f:
        f.write(insert + '\n')
```