Iteración 2

Sistemas Transaccionales

Juan David Guevara Arévalo - 202116875

Daniela Parra Martinez - 202013036

Kevin David Alvarez Romero - 202022834

k.alvarezr

1. Análisis

Teniendo en cuenta los nuevos requerimientos en esta iteración 2 nuestros diagramas de UML y Relacional son acordes con estos requerimientos sin embargo algunos requieren de la creación de una nueva tabla “Consumo”. Esta tabla estará relacionada con un servicio o producto, además estará relacionada con la tabla reserva, lo que permitirá unir los consumos con usuarios, servicio. Esto con el fin de poder realizar todos los nuevos requerimientos. (Ver diagrama al final del documento).

2. Proceso de la carga de datos

Para el proceso de la carga de datos decidimos elegir la opción de crear un programa de generación automática de datos. Para esto usamos la libreria ***faker*** la cual crea datos random de fechas, paises, nombres, etc. En total tuvimos aproximadamente 600 mil registros cargados.

3. Diseño de la aplicación

RFC1: MOSTRAR EL DINERO RECOLECTADO POR SERVICIOS EN CADA HABITACIÓN EN EL ÚLTIMO AÑO CORRIDO.

Para resolver este requerimiento, proponemos crear un índice secundario B + sobre la columna “fecha” de la tabla “consumos”. El motivo para elegir este tipo de índice es el tipo de búsqueda que requiere la consulta, al ser una consulta sobre fecha, es más

eficiente tener un índice B+ para obtener los registros en el rango del último año, sin este índice, sería necesario recorrer toda la tabla de consumos, lo cual puede ser extremadamente ineficiente debido al gran volumen de datos, en especial, en esta tabla.

RFC2: MOSTRAR LOS 20 SERVICIOS MÁS POPULARES EN UN RANGO DE FECHAS

Para resolver este requerimiento no es necesario crear un nuevo índice. En este caso el motivo es que se propone usar el mismo índice creado para resolver el requerimiento funcional 1, pues, la principal tabla necesaria para la consulta, de nuevo, es la tabla de consumos, la cual permite relacionar un consumo con un servicio, y lo más importante, permite filtrar los consumos para obtener solo los que se encuentren en el rango, es por eso que se considera que el índice secundario B + creado anteriormente es ideal para trabajar este requerimiento.

RFC3: MOSTRAR EL INDICE DE OCUPACIÓN EN CADA UNA DE LAS HABITACIONES DEL HOTEL.

Asumimos que es dado el id del hotel

Para resolver este requerimiento, se decidió crear un índice secundario B + para la columna “inicio” de la tabla de “reservas”. La razón es similar a lo que sucede en el requerimiento 1, pues, en este caso, se toman todas las habitaciones de un hotel, y por cada una de ellas, se revisan las reservas que hayan comenzado en el último año. Sin el índice, se requeriría recorrer todos los registros de la tabla reservas buscando los registros que cumplan con la condición, sin embargo, al añadir el índice, el manejador puede hacer un “INDEX RANGE SCAN”. Esto ayuda a reducir considerablemente la cantidad de reservas consultadas, en especial si hay registros en un rango muy amplio. Adicionalmente, se decidió crear un índice B+ sobre la columna “idhabitacion” de la tabla “reservas”

RFC4: MOSTRAR LOS SERVICIOS QUE CUMPLEN CON CIERTA CARACTERÍSTICA

Para resolver este requerimiento, se decidio crear un indice secundario B + en la columna “cobro” de la tabla servicios y en la columna “tipo_cobro”de la tabla servicios ademas de la columna “horario” de la tabla reserva servicios. La razón del porque se crearon estos indice es que se utilizan para optimizar la consulta que filtran y ordenan por cobro y tipo de cobro del servicio para filtrarlos servicios que cumplen con una determinada caracteristica.

RFC5 - MOSTRAR EL CONSUMO EN HOTELANDES POR UN USUARIO DADO, EN UN RANGO DE FECHAS INDICADO.

Sí, es necesario crear un índice para mejorar el rendimiento de la consulta básica.

La consulta básica selecciona los datos de un usuario específico, en un rango de fechas de un año. La consulta debe realizar una búsqueda en la tabla consumos para encontrar todos los consumos realizados por el usuario especificado, en el rango de fechas especificado. El número de registros en la tabla consumos puede ser muy grande, por lo que la consulta puede ser muy lenta.

Para mejorar el rendimiento de esta consulta, podemos crear un índice en la columna id_usuario de la tabla consumos. Este índice permitirá a la base de datos encontrar rápidamente todos los consumos realizados por el usuario especificado.

En el caso de la columna id_usuario de la tabla consumos, la selectividad es alta. Esto se debe a que cada consumo está asociado con un único usuario. Por lo tanto, crear un índice en esta columna mejorará significativamente el rendimiento de la consulta básica.

El tipo de índice más adecuado para esta situación es un índice B+. Los índices B+ son eficientes para consultas de rango, que es el tipo de consulta que se realiza para consultar el consumo de un usuario específico en un rango de fechas.

RFC6:ANALIZAR LA OPERACIÓN DE HOTELANDES

Para resolver este requerimiento, se decidió crear un índice simple en la columna “fecha_reservas” de la tabla reservas. La razón del porqué se creó este índice es que se utiliza para optimizar la consulta que filtra y ordena por la fecha de reserva para saber la mayor ocupación y la menor demanda. Además, se creó un índice compuesto en las columnas “fecha y costo” de la tabla de consumos. La razón de que se use el índice para optimizar la consulta que filtra y ordena por ambas columnas para saber los mayores ingresos de HotelAndes.

RFC7 - ENCONTRAR LOS BUENOS CLIENTES

Para mejorar el rendimiento de esta consulta, podemos crear índices en las siguientes columnas:

- idreserva: Esta columna es la clave primaria de la tabla reservas, por lo que su selectividad es del 100%. Un índice en esta columna permitirá a la base de datos encontrar rápidamente todas las reservas que cumplan con los criterios especificados.
- idhabitacion: Esta columna es una clave foránea de la tabla reservas, por lo que su selectividad es alta. Un índice en esta columna permitirá a la base de datos encontrar rápidamente todas las reservas que se realizaron en una habitación específica.
- iddotacion: Esta columna es una clave foránea de la tabla hdotaciones, por lo que su selectividad es alta. Un índice en esta columna permitirá a la base de datos encontrar rápidamente todas las reservas que incluyen una dotación específica.

En el caso de las columnas idreserva, idhabitacion y iddotacion, la selectividad es alta. Esto se debe a que cada reserva está asociada con un único registro en cada tabla. Por lo tanto, crear un índice en estas columnas mejorará significativamente el rendimiento de la consulta básica.

El tipo de índice más adecuado para esta situación es un índice B+. Los índices B+ son eficientes para consultas de rango, que es el tipo de consulta que se realiza para

consultar los datos de los clientes que han realizado una estadía de al menos dos semanas o un consumo superior a $15.000.000 en el último año.

RFC8: ENCONTRAR LOS SERVICIOS QUE NO TIENEN MUCHA DEMANDA

Para resolver este requerimiento, se decidió crear un índice simple en la columna “horario” de la tabla reserva_servicios. La razón del porqué se creó este índice es que se utiliza para optimizar la consulta que filtra y ordenan por el horario de reserva para la búsqueda seleccionar los servicios que no tienen mucha demanda.

RFC9 - CONSULTAR CONSUMO EN HOTELANDES

Desde el punto de vista de la selectividad:

- La condición consumos.fecha BETWEEN ADD_MONTHS(CURRENT_DATE, -12) AND CURRENT_DATE se refiere a un rango de fechas dentro de los últimos 12 meses. Esto podría ser selectivo si hay una cantidad significativa de datos en la tabla consumos y la mayoría de las filas se encuentran dentro de ese rango.
- La condición consumos.id_servicio = 1 es específica para el id_servicio igual a 1. La selectividad dependerá de cuántas filas en la tabla consumos tienen un id_servicio igual a 1.

Dado el análisis, aquí hay consideraciones sobre la necesidad de índices:

- Para la tabla usuarios, si la tabla es grande y la consulta se ejecuta con frecuencia, un índice en la columna id sería útil, ya que la condición INNER JOIN se basa en este campo.
- Para la tabla consumos, si la tabla es grande y la mayoría de las filas tienen un id_servicio igual a 1, un índice en la columna id_servicio sería útil.

En cuanto al tipo de índice:

- Para el id en la tabla usuarios, un índice secundario sería apropiado, ya que no es una clave principal, pero se utiliza para unir las tablas.
- Para la columna id_servicio en la tabla consumos, un índice secundario también sería apropiado si la condición es selectiva.

RFC10 - CONSULTAR CONSUMO EN HOTELANDES – RFC9-V2

Se aplica la misma logica que para el requerimiento 9

RFC11: CONSULTAR FUNCIONAMIENTO

No se creo ningun indice para este requerimiento ya que en esta consulta se tiene que recorrer toda la tabla de consumos y reservas para poder obtener el funcionamiento.

3.1. Diseño de consultas

RFC1:

```sql
-- REQ. 1
create index consumos_fecha_idx on consumos(fecha);

select * from (
select rownum as r, idhabitacion, coste_total from (
select sum(consumos.costo) as coste_total, habitaciones.idhabitacion from consumos
    inner join reservas on reservas.idreserva = consumos.id_reserva
    inner join habitaciones on habitaciones.idhabitacion = reservas.idhabitacion
where consumos.fecha ⩾ add_months(CURRENT_DATE, -12)
group by habitaciones.idhabitacion
order by coste_total desc
)) where r between (:page - 1) * 20 and :page * 20;
```

Operation	Params	Rows	Total Cost
Select		70907	81.0
Filter			
Access (VIEW)		70907	81.0
Transformation (COUNT)			
Access (VIEW)		70907	81.0
Order By (SORT ORDER BY)		70907	81.0
Group By (HASH GROUP BY)		70907	81.0
Nested Loops		70907	76.0
Nested Loops		70907	76.0
Full Scan (TABLE ACCESS FULL)	table: CONSUMOS;	70907	73.0
Unique Index Scan (INDEX UNIQUE SC/	index: RESERVAS_PK;	1	0.0
Index Scan (TABLE ACCESS BY INDEX RO\	table: RESERVAS;	1	0.0

RFC2:

```sql
-- REQ. 2
select servicios.id, servicios.nombre, count(consumos.id) as cantidad_consumos from servicios
    inner join consumos on consumos.id_servicio = servicios.id
where consumos.fecha between :date1 and :date2
group by servicios.id, servicios.nombre
order by cantidad_consumos desc
fetch first 20 rows only;
```

Operation	Params	Rows	Total Cost
Select		20	78.0
Order By (SORT ORDER BY)		20	78.0
Access (VIEW)		20	77.0
Unknown (WINDOW SORT PUSHED RANK)		181	77.0
Group By (HASH GROUP BY)		181	77.0
Filter			
Hash Join		181	75.0
Full Scan (TABLE ACCESS FULL)	table: CONSUMOS;	181	71.0
Full Scan (TABLE ACCESS FULL)	table: SERVICIOS;	800	4.0

RFC3:

```sql
-- REQ. 3
create index reservas_fecha_idx on reservas(inicio);
create index reservas_hab_idx on reservas(idhabitacion);

select
    habitaciones.idhabitacion,
    ROUND((sum(reservas.cantidadpersonas) / sum(habitaciones.capacidad)) * 100, 2) as ocupacion,
    count(reservas.idreserva) as cantidad_reservas
from habitaciones
    inner join reservas on reservas.idhabitacion = habitaciones.idhabitacion
where idhotel = :idhotel and reservas.inicio ⩾ ADD_MONTHS(CURRENT_DATE, -12)
group by habitaciones.idhabitacion
order by ocupacion desc;
```

Operation	Params	Rows	Total Cost
Select		384	9.0
Order By (SORT ORDER BY)		384	9.0
Sort (SORT GROUP BY NOSORT)		384	9.0
Nested Loops		384	8.0
Nested Loops		416	8.0
Index Scan (TABLE ACCESS BY INDEX ROWID)	table: HABITACIONES;	4	1.0
Full Index Scan (INDEX FULL SCAN)	index: HABITACIONES_PK;	423	1.0
Index Scan (INDEX RANGE SCAN)	index: RESERVAS_HAB_IDX;	104	1.0
Index Scan (TABLE ACCESS BY INDEX ROWID)	table: RESERVAS;	91	2.0

RFC4:

```
SELECT s.id, s.nombre, s.capacidad, s.tipo_cobro, s.cobro
FROM servicios s, reserva_servicios rs
WHERE s.cobro BETWEEN 50 AND 200
      AND rs.horario BETWEEN TO_DATE('2022-11-01', 'YYYY-MM-DD') AND TO_DATE('2022-11-10', 'YYYY-MM-DD')
      AND s.tipo_cobro = 'DIA';
```

```
CREATE INDEX idx_servicios_tipo_cobro ON servicios(tipo_cobro);
CREATE INDEX idx_reserva_servicios_horario ON reserva_servicios(horario);
CREATE INDEX idx_servicios_cobro ON serviciosl(cobro);
```

SQL | 0,032 segundos

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	0
MERGE JOIN		CARTESIAN	1	0
TABLE ACCESS	SERVICIOS	BY INDEX ...	1	0
Filter Predicates				
S.TIPO_COBRO='DIA'				
INDEX	IDX_SERVICIOS_COBRO	RANGE SCAN	1	0
Access Predicates				
AND				
S.COBRO>=50				
S.COBRO<=200				
BUFFER		SORT	1	0
INDEX	IDX_RESERVA_SERVICIOS_HORARIO	RANGE SCAN	1	0
Access Predicates				
AND				
RS.HORARIO>=TO_DATE(' 2022-11-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				
RS.HORARIO<=TO_DATE(' 2022-11-10 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				

RFC5:

```
select usuarios.id, usuarios.nombre, consumos.costo
from usuarios join consumos on consumos.id_usuario = usuarios.id
where consumos.fecha between add_months(current_date, -12) and current_date and usuarios.id = 1;
```

PLAN_TABLE_OUTPUT

```
Plan hash value: 3175816248

-------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |             |     1 |   150 |     2   (0)| 00:00:01 |
|*  1 |  FILTER                       |             |       |       |            |          |
|   2 |   NESTED LOOPS                |             |     1 |   150 |     2   (0)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID| USUARIOS    |     1 |   115 |     0   (0)| 00:00:01 |
|*  4 |     INDEX UNIQUE SCAN         | USUARIO_PK  |     1 |       |     0   (0)| 00:00:01 |
|*  5 |    TABLE ACCESS FULL          | CONSUMOS    |     1 |    35 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(CURRENT_DATE>=ADD_MONTHS(CURRENT_DATE,(-12)))
   4 - access("USUARIOS"."ID"=1)
   5 - filter("CONSUMOS"."ID_USUARIO"=1 AND "CONSUMOS"."FECHA"<=CURRENT_DATE AND
              "CONSUMOS"."FECHA">=ADD_MONTHS(CURRENT_DATE,(-12)))

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

RFC6:

```
(SELECT 'Mayor Ocupación' AS tipo, fecha_reserva AS fecha, COUNT(*) AS valor
FROM reservas
GROUP BY fecha_reserva
ORDER BY COUNT(*) DESC
FETCH FIRST 1 ROW ONLY)
UNION
(SELECT 'Mayores Ingresos' AS tipo, fecha, SUM(costo) AS valor
FROM consumos
GROUP BY fecha
ORDER BY SUM(costo) DESC
FETCH FIRST 1 ROW ONLY)
UNION
(SELECT 'Menor Demanda' AS tipo, fecha_reserva AS fecha, COUNT(*) AS valor
FROM reservas
GROUP BY fecha_reserva
ORDER BY COUNT(*) ASC
FETCH FIRST 1 ROW ONLY);

CREATE INDEX consumos_fecha_costo ON consumos(fecha, costo);
CREATE INDEX reservas_fecha ON reservas(fecha_reserva);
```

SQL | 0,121 segundos

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			3	6
SORT		UNIQUE	3	6
UNION-ALL				
VIEW	SYS.null		1	2
Filter Predicates				
from$_subquery$_002.rowlimit_$$_rownumber<=1				
WINDOW		SORT PUS...	1	2
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC)<=1				
SORT		GROUP BY ...	1	2
INDEX	RESERVAS_FECHA	FULL SCAN	1	1
VIEW	SYS.null		1	2
Filter Predicates				
from$_subquery$_004.rowlimit_$$_rownumber<=1				
WINDOW		SORT PUS...	1	2
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY SUM(COSTO) DESC)<=1				
SORT		GROUP BY ...	1	2
INDEX	CONSUMOS_FECHA_COSTO	FULL SCAN	1	1
VIEW	SYS.null		1	2
Filter Predicates				
from$_subquery$_006.rowlimit_$$_rownumber<=1				
WINDOW		SORT PUS...	1	2
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*))<=1				
SORT		GROUP BY ...	1	2
INDEX	RESERVAS_FECHA	FULL SCAN	1	1

RFC7:

```
SELECT
u.nombre AS nombre_cliente,
r.inicio AS fecha_inicio_estadia,
r.fin AS fecha_fin_estadia,
SUM(d.costoadicional) AS consumo_total
FROM usuarios u
JOIN registros rg ON u.numdocumento = rg.docpersona
JOIN reservas r ON rg.idreserva = r.idreserva
JOIN habitaciones h ON r.idhabitacion = h.idhabitacion
JOIN hdotaciones hd ON h.idhabitacion = hd.idhabitacion
JOIN dotaciones d ON hd.iddotacion = d.iddotacion
WHERE r.inicio >= add_months(CURRENT_DATE, -12)
GROUP BY u.nombre, r.inicio, r.fin
HAVING (SUM(EXTRACT(DAY FROM r.fin - r.inicio)) >= 14 OR SUM(d.costoadicional) > 15000000)
ORDER BY nombre_cliente, fecha_inicio_estadia;
```

Por falta de tiempo esta consulta no termino de correr, por lo que no hay plan de ejecucion

RFC8:

```
SELECT s.id, s.nombre, s.capacidad, s.tipo_cobro, s.cobro
FROM servicios s
WHERE s.id NOT IN (
    SELECT rs.id_salon
    FROM reserva_servicios rs
    WHERE rs.horario >= ADD_MONTHS(SYSDATE, -12) -- Obtener fecha hace un año
    GROUP BY rs.id_salon
    HAVING COUNT(DISTINCT TO_CHAR(rs.horario, 'IW')) >= 3
);

CREATE INDEX idx_horario ON reserva_servicios(horario);
```

SQL | 0,055 segundos

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	5
FILTER				
Filter Predicates				
NOT EXISTS (SELECT 0 FROM (SELECT TO_CHAR(INTERNAL_FUNCTION(RS.HORARIO),'IW') $vm_col_1,RS.ID_SPA $vm_col_2 FROM RESERVA_SERVIC				
TABLE ACCESS	SERVICIOS	FULL	1	4
FILTER				
Filter Predicates				
AND				
LNNVL($vm_col_2<>:B1)				
COUNT($vm_col_1)>=3				
HASH		GROUP BY	1	1
VIEW	SYS.VM_NWVW_1		1	1
SORT		GROUP BY	1	1
TABLE ACCESS	RESERVA_SERVICIOS	BY INDEX ...	1	0
INDEX	IDX_RESERVA_SERVICIOS_HORARIO	RANGE SCAN	1	0
Access Predicates				
RS.HORARIO>=ADD_MONTHS(SYSDATE@!,(-12))				

RCF9:

```
select usuarios.id, count(consumos.id_servicio) from usuarios
inner join consumos on consumos.id_usuario = usuarios.id
where consumos.fecha between add_months(current_date, -12) and current_date and consumos.id_servicio = 1
having count(consumos.id_servicio)>0
group by usuarios.id;
```

PLAN_TABLE_OUTPUT

```
Plan hash value: 3502741118

-------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |     1 |    35 |     3  (34)| 00:00:01 |
|*  1 |  HASH GROUP BY      |          |     1 |    35 |     3  (34)| 00:00:01 |
|*  2 |   FILTER            |          |       |       |            |          |
|*  3 |    TABLE ACCESS FULL| CONSUMOS |     1 |    35 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(COUNT("CONSUMOS"."ID_SERVICIO")>0)
   2 - filter(CURRENT_DATE>=ADD_MONTHS(CURRENT_DATE,(-12)))
   3 - filter("CONSUMOS"."ID_USUARIO" IS NOT NULL AND
              "CONSUMOS"."ID_SERVICIO"=1 AND "CONSUMOS"."FECHA"<=CURRENT_DATE AND
              "CONSUMOS"."FECHA">=ADD_MONTHS(CURRENT_DATE,(-12)))

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

RCF10:

```
select usuarios.id, count(consumos.id_servicio) from usuarios
inner join consumos on consumos.id_usuario = usuarios.id
where consumos.fecha between add_months(current_date, -12) and current_date and consumos.id_servicio = 1
having count(consumos.id_servicio)=0
group by usuarios.id;
```

PLAN_TABLE_OUTPUT

```
Plan hash value: 3502741118

-----------------------------------------------------------------------------------
| Id  | Operation           | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |           |     1 |    35 |     3  (34)| 00:00:01 |
|*  1 |  HASH GROUP BY      |           |     1 |    35 |     3  (34)| 00:00:01 |
|*  2 |   FILTER            |           |       |       |            |          |
|*  3 |    TABLE ACCESS FULL| CONSUMOS  |     1 |    35 |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(COUNT("CONSUMOS"."ID_SERVICIO")=0)
   2 - filter(CURRENT_DATE>=ADD_MONTHS(CURRENT_DATE,(-12)))
   3 - filter("CONSUMOS"."ID_USUARIO" IS NOT NULL AND
              "CONSUMOS"."ID_SERVICIO"=1 AND "CONSUMOS"."FECHA"<=CURRENT_DATE AND
              "CONSUMOS"."FECHA">=ADD_MONTHS(CURRENT_DATE,(-12)))

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

RFC11:

```
select
   c.SEMANA,
   servicio_menos_consumo,
   menor_cantidad,
   servicio_mas_consumo,
   mayor_cantidad,
   habitacion_menos_pedida,
   menor_cantidad_reservas,
   habitacion_mas_pedida,
   mayor_cantidad_reservas
from (
   select
      semana,
```

```
       max(id_servicio) keep ( dense_rank first order by consumos_servicio) as servicio_menos_consumo,
       min(consumos_servicio) as menor_cantidad,
       max(id_servicio) keep ( dense_rank last order by consumos_servicio) as servicio_mas_consumo,
       max(consumos_servicio) as mayor_cantidad
   from (
       select TO_CHAR(consumos.fecha, 'WW') as SEMANA, consumos.id_servicio, count(consumos.id_servicio) as consumos_servicio
from consumos
       where consumos.id_servicio IS NOT NULL and EXTRACT(year from consumos.fecha) = EXTRACT(year from CURRENT_DATE)
       group by TO_CHAR(consumos.fecha, 'WW'), consumos.id_servicio
   )
   group by semana
) c full join (
   select
       semana,
       max(idhabitacion) keep ( dense_rank first order by reservas_habitacion) as habitacion_menos_pedida,
       min(reservas_habitacion) as menor_cantidad_reservas,
       max(idhabitacion) keep ( dense_rank last order by reservas_habitacion) as habitacion_mas_pedida,
       max(reservas_habitacion) as mayor_cantidad_reservas
   from (
       select TO_CHAR(reservas.fecha_reserva, 'WW') as SEMANA, reservas.idhabitacion, count(reservas.idhabitacion) as
reservas_habitacion from reservas
       where reservas.idhabitacion IS NOT NULL and EXTRACT(year from reservas.fecha_reserva) = EXTRACT(year from CURRENT_DATE)
       group by TO_CHAR(reservas.fecha_reserva, 'WW'), reservas.idhabitacion
   )
   group by semana
) h on c.SEMANA = h.SEMANA;
```

Operation	Params	Rows	Total Cost
Select		2	8.0
Access (VIEW)		2	8.0
Hash Join (HASH JOIN FULL OUTER)		2	8.0
Access (VIEW)		1	4.0
Group By (SORT GROUP BY)		1	4.0
Access (VIEW)		1	3.0
Group By (HASH GROUP BY)		1	3.0
Full Scan (TABLE ACCESS FULL)	table: CONSUMOS;	1	2.0
Access (VIEW)		1	4.0
Group By (SORT GROUP BY)		1	4.0
Access (VIEW)		1	3.0
Group By (HASH GROUP BY)		1	3.0
Full Scan (TABLE ACCESS FULL)	table: RESERVAS;	1	2.0

Pruebas de ejecución

-Se agregaron datos aleatorios y acordes con la base de datos para realizar las pruebas de ejecución

RFC1:

Consulta:

R	IDHABITACION	COSTE_TOTAL
1	10363	102209637
2	23694	97187810
3	3470	94059983
4	91620	92406313
5	42460	91828417
6	8202	90431539
7	93081	89996984
8	91951	89593782
9	46246	89459559
10	81038	89224133
11	95202	88378524
12	1036	88073027
13	46524	87878518
14	38322	87856239
15	91307	87735173
16	81603	87578759
17	108944	87479705
18	12475	87313785
19	24578	87171711
20	83983	86817576

Interfaz:

Dinero Recolectado en servicios por cada Habitación

Anterior Pagina Pagina 1 Siguiente

#	ID Habitación	Dinero recolectado
1	10363	102209637
2	23694	97187810
3	3470	94059983
4	91620	92406313
5	42460	91828417
6	8202	90431539
7	93081	89996984
8	91951	89593782
9	46246	89459559
10	81038	89224133

RFC4:

	ID	NOMBRE	CAPACIDAD	TIPO_COBRO	COBRO
1	1	Servicio 1	10	DIA	50

RFC6:

	TIPO	FECHA	VALOR
1	Mayor Ocupación	06/10/23	4
2	Mayores Ingresos	24/10/23	432
3	Menor Demanda	05/10/23	2

RFC8:

ID	NOMBRE	CAPACIDAD	TIPO_COBRO	COBRO
1	Servicio 1	10	DIA	50
2	Servicio 2	5	HORA	20
3	Servicio 3	15	UNICO	100

Diagrama UML :