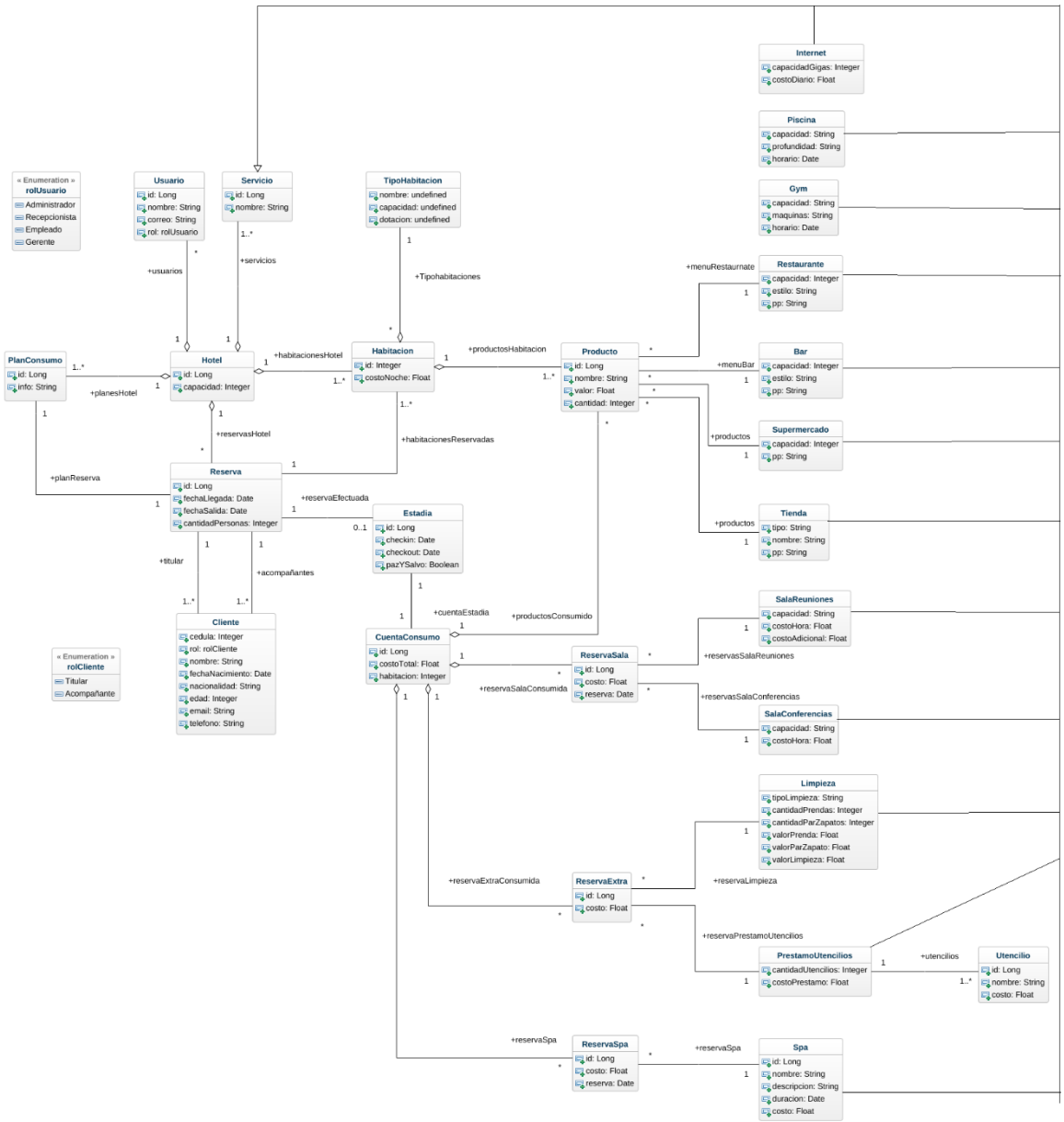
Informe proyecto SISTRANS iteración 2

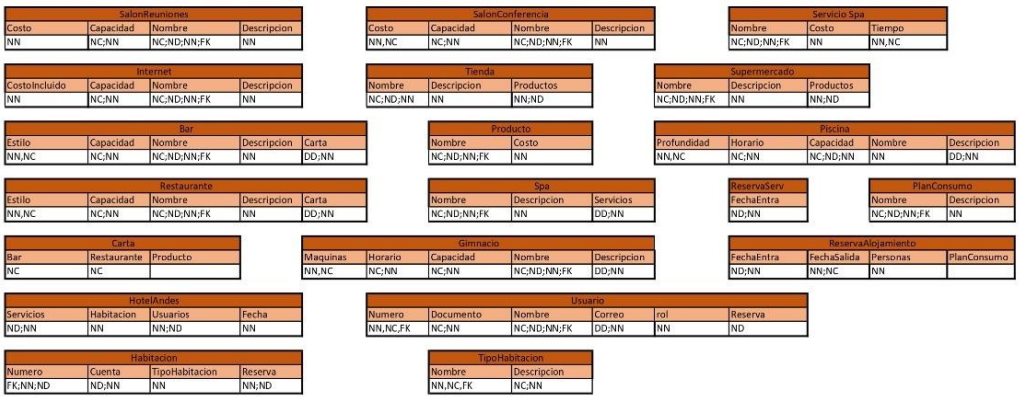
iteración 1:

UML



E/R

Tablas



SalonReuniones			
Costo	Capacidad	Nombre	Descripcion
NN	NC;NN	NC;ND;NN;FK	NN

SalonConferencia			
Costo	Capacidad	Nombre	Descripcion
NN,NC	NC;NN	NC;ND;NN;FK	NN

Servicio Spa		
Nombre	Costo	Tiempo
NC;ND;NN;FK	NN	NN,NC

Internet			
CostoIncluido	Capacidad	Nombre	Descripcion
NN	NC;NN	NC;ND;NN;FK	NN

Tienda		
Nombre	Descripcion	Productos
NC;ND;NN	NN	NN;ND

Supermercado		
Nombre	Descripcion	Productos
NC;ND;NN;FK	NN	NN;ND

Bar				
Estilo	Capacidad	Nombre	Descripcion	Carta
NN,NC	NC;NN	NC;ND;NN;FK	NN	DD;NN

Producto	
Nombre	Costo
NC;ND;NN;FK	NN

Piscina				
Profundidad	Horario	Capacidad	Nombre	Descripcion
NN,NC	NC;NN	NC;ND;NN	NN	DD;NN

Restaurante				
Estilo	Capacidad	Nombre	Descripcion	Carta
NN,NC	NC;NN	NC;ND;NN;FK	NN	DD;NN

Spa		
Nombre	Descripcion	Servicios
NC;ND;NN;FK	NN	DD;NN

ReservaServ	
FechaEntra	
ND;NN	

PlanConsumo	
Nombre	Descripcion
NC;ND;NN;FK	NN

Carta		
Bar	Restaurante	Producto
NC	NC	

Gimnacio				
Maquinas	Horario	Capacidad	Nombre	Descripcion
NN,NC	NC;NN	NC;NN	NC;ND;NN;FK	DD;NN

ReservaAlojamiento			
FechaEntra	FechaSalida	Personas	PlanConsumo
ND;NN	NN;NC	NN	

HotelAndes			
Servicios	Habitacion	Usuarios	Fecha
ND;NN	NN	NN;ND	NN

Usuario					
Numero	Documento	Nombre	Correo	rol	Reserva
NN,NC,FK	NC;NN	NC;ND;NN;FK	DD;NN	NN	ND

Habitacion			
Numero	Cuenta	TipoHabitacion	Reserva
FK;NN;ND	ND;NN	NN	NN;ND

TipoHabitacion	
Nombre	Descripcion
NN,NC,FK	NC;NN

Normalización

ROLUSUARIOS

1FN (Primera Forma Normal): Es necesario que los atributos de la tabla sean atómicos, para esto, cada uno de los atributos empleados: ID, NOMBRE y DESCRIPCION, cumplen con esa condición. Por lo tanto, si lo está.

2FN (Segunda Forma Normal): Para este caso, no puede haber dependencias parciales. Por el lado de los atributos. Ni NOMBRE ni DESCRIPCIÓN, tienen esta condición. Pues ambos dependen únicamente de la PK de la tabla.

3FN (Tercera Forma Normal): Similarmente, NOMBRE y DESCRIPCIÓN dependen enteramente de ID como PK. De esta forma, ninguno funciona como relación transitiva de algún atributo diferente. Por lo tanto, se cumple con esta última condición.

USUARIOS

1FN (Primera Forma Normal): Todos los atributos son de índole atómico. De esta forma, se encuentra en 1FN.

2FN (Segunda Forma Normal): NUMERO DOCUMENTO como PK, se encarga de determianr los atributos restantes, impidiendo la creación de dependencias parciales, se encuentra en 2FN.

3FN (Tercera Forma Normal): Al depender TIPO_USUARIO de la columna ID de la tabla TIPOS_USUARIO, como llave primaria de esa tabla, asegura en últimas la imposibilidad de dependencias transitivas de los atributos. Se encuentra, de esta forma en 3FN.

TIPOSHABITACIONES

1FN (Primera Forma Normal): No hay atributos repetidos, ni cada atributo cuenta con multivalor, al ser todos atómicos, se encuentra en 1FN.

2FN (Segunda Forma Normal): Para esta ocasión, la clave primaria ID, se encarga de determinar los demás atributos. De esta forma, no hay ningún tipo de relaciones parciales mediante los atributos, asegurando la 2FN.

3FN (Tercera Forma Normal): Para los atributos, cada atributo no primo, depende de manera directa de un atributo que si es primo. De esta forma, no hay formación de relaciones transitivas dentro de la tabla en cuestión.

HABITACIONES

1FN (Primera Forma Normal): Al analizar cada atributo, se caracteriza por poseer valores atómicos, convirtiendo la tabla en 1FN.

2FN (Segunda Forma Normal): Cumple con la 2FN, ya que la clave primaria NUMERO determina completamente todas las demás columnas. No hay dependencias parciales.

3FN (Tercera Forma Normal): No hay dependencias transitivas en esta tabla. La columna TIPO_HABITACION depende directamente de la clave primaria NUMERO.

PLANESCONSUMO

1FN (Primera Forma Normal): No hay valores repartidos, cada atributo es atómico. En conclusión, cumple la 1FN.

2FN (Segunda Forma Normal): La tabla también cumple con la 2FN, ya que la clave primaria ID determina completamente todas las demás columnas. No hay dependencias parciales.

3FN (Tercera Forma Normal): En esta tabla, no hay dependencias transitivas. Cada atributo no clave depende directamente de la clave primaria ID.

RESERVACIONES

1FN (Primera Forma Normal): La tabla cumple con la 1FN, ya que no hay valores repetidos ni listas de valores en ninguna celda. Cada fila tiene valores únicos para ID_HABITACION, TITULAR, PLAN_ELEGIDO, TARIFA_TOTAL, FECHA_LLEGADA y FECHA_SALIDA.

2FN (Segunda Forma Normal): Cumple con la 2FN, ya que la clave primaria ID_HABITACION determina completamente todas las demás columnas. No hay dependencias parciales.

3FN (Tercera Forma Normal): No hay dependencias transitivas en esta tabla. Todos los atributos no clave dependen directamente de la clave primaria ID_HABITACION.

ESTADIAS

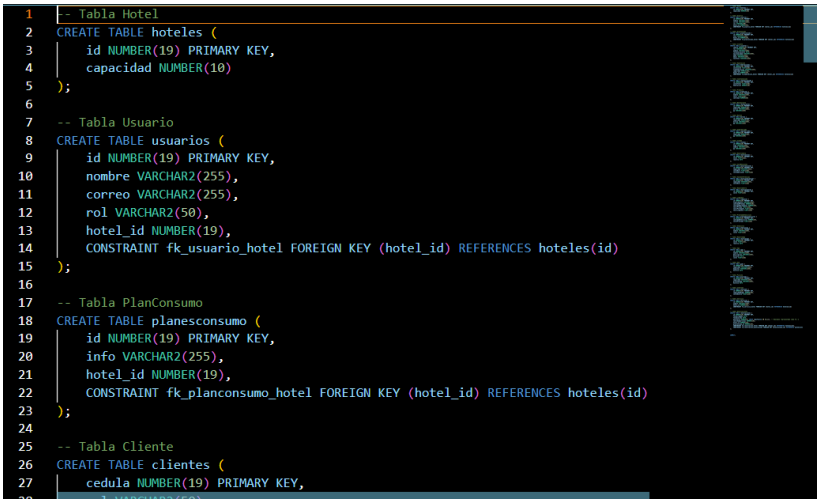
1FN (Primera Forma Normal): No se encuentran valores repetidos y cada uno de los atributos es atómico.

2FN (Segunda Forma Normal): La tabla también cumple con la 2FN, ya que la clave primaria LlegadaClienteID se encarga de determinar las demás columnas. Asegurando la no existencia de relaciones aprciales.

3FN (Tercera Forma Normal): No se evidencia transitividad de relaciones, pues cada atributo depende de forma única de la PK de la tabla.

SQL Tablas

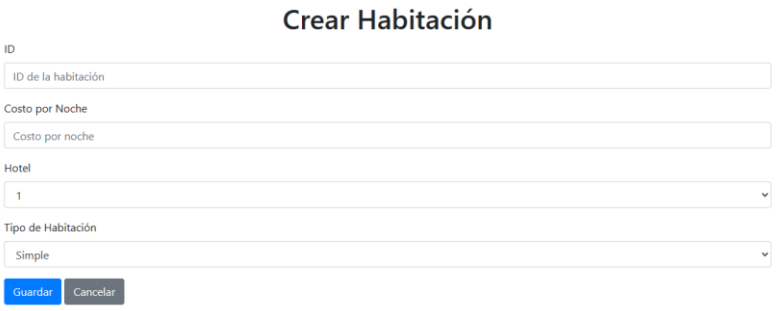
Adjuntamos el archivo .sql de las tablas



```sql
-- Tabla Hotel
CREATE TABLE hoteles (
    id NUMBER(19) PRIMARY KEY,
    capacidad NUMBER(10)
);

-- Tabla Usuario
CREATE TABLE usuarios (
    id NUMBER(19) PRIMARY KEY,
    nombre VARCHAR2(255),
    correo VARCHAR2(255),
    rol VARCHAR2(50),
    hotel_id NUMBER(19),
    CONSTRAINT fk_usuario_hotel FOREIGN KEY (hotel_id) REFERENCES hoteles(id)
);

-- Tabla PlanConsumo
CREATE TABLE planesconsumo (
    id NUMBER(19) PRIMARY KEY,
    info VARCHAR2(255),
    hotel_id NUMBER(19),
    CONSTRAINT fk_planconsumo_hotel FOREIGN KEY (hotel_id) REFERENCES hoteles(id)
);

-- Tabla Cliente
CREATE TABLE clientes (
    cedula NUMBER(19) PRIMARY KEY,
```

Requerimientos funcionales

Todos los requerimientos funcionales fueron logrados para la iteración 1 adjuntamos algunas fotos y también el día de la sustentación se evidenciara cada uno de los requerimientos.



Así para cada requerimiento y es totalmente funcional es decir los métodos CRUD para cada requerimiento fueron cubiertos en su totalidad

Requerimientos no funcionales

a- Persistencia

En cuanto a la persistencia evidenciamos que cuando creamos un nuevo registro en la base de datos ya sea desde el html o desde una sentencia SQL esta se mantiene mientras que no sea eliminada usando el ejemplo anterior de las fotos de la tabla de habitaciones, todos los registros persisten en la BD:

ID	COSTONOCHE	HOTEL_ID	TIPO_HABITACION_ID
1	74.24	1	1
2	178.39	2	2
3	88.94	3	3
4	190.1	1	1
5	137.96	2	2
6	130.24	3	3
7	66.39	1	1
8	72.3	2	2
9	152.47	3	3
10	111.71	1	1

Page 1 of 50 (1-10 of 500 rows)

b- Centralizada

En cuanto a si nuestra base de datos esta centralizada la respuesta es si debido a que tenemos integridad en los datos y eficiencia esto lo podemos ver cuando hacemos sentencias como las de SELECT, eliminar por id o actualizar un registro, además la relaciones FK y PK de las tablas están todas correctas no pueden haber 2 mismas PK en una misma tabla, en las otras tablas solo puede haber una única FK asociada a cada registro etc.

Ejemplo tenemos la tabla reservaciones que tiene FK de cuenta consumo, id cliente, etc. Y si nos damos cuenta no se repite ningún valor solo se puede repetir el de id cliente lo cual es correcto porque un mismo cliente puede tener varias reservas

ID	FECHALLEGADA	FECHASALIDA	CANTIDADPERSONAS	HOTEL_ID	PLANCONSUMO_ID	TIPOHABITACION_ID	TITULAR_ID	HABITACIOI
306	30-JAN-24	14-FEB-24	2	1	1	1	100040935	
307	28-NOV-23	01-DEC-23	1	2	1	1	100040587	
1	03-SEP-24	13-SEP-24	2	2	2	2	100040529	
308	31-AUG-24	01-SEP-24	3	3	2	1	100040520	
2	13-JAN-24	27-JAN-24	1	2	2	3	100041227	
3	17-JUL-24	19-JUL-24	3	3	1	2	100041077	
4	02-FEB-24	10-FEB-24	3	1	2	3	100041221	
5	06-MAR-24	09-MAR-24	2	2	1	3	100041439	
6	24-NOV-23	05-DEC-23	2	2	2	1	100041143	
7	23-FEB-24	27-FEB-24	2	2	1	3	100040798	

Page 1 of 50 (1-10 of 500 rows)

SQL Requerimientos

Además de tener darse cuenta de que los requerimientos son funcionales en la sustentación podemos rectificar las sentencias sql de cada requerimiento en el repositorio

Donde cada repositorio esta con sus sentencias sql debidas y siguiendo la arquitectura de SW demandada

```java
public interface HabitacionRepository extends JpaRepository<Habitacion, Long> {

    @Query(value = "SELECT * FROM habitaciones", nativeQuery = true)
    Collection<Habitacion> darHabitaciones();

    @Modifying
    @Transactional
    @Query(value = "INSERT INTO habitaciones (id, COSTONOCHE, HOTEL_ID, TIPO_HABITACION_ID)
    void insertarHabitacion(@Param("id") Integer integer, @Param("costoNoche") Double costol

    @Modifying
    @Transactional
    @Query(value = "UPDATE habitaciones SET COSTONOCHE = :costoNoche, HOTEL_ID = :hotel_id,
    void actualizarHabitacion(@Param("id") Long id, @Param("costoNoche") Double costoNoche,

    @Modifying
    @Transactional
    @Query(value = "DELETE FROM habitaciones WHERE id = :id", nativeQuery = true)
    void eliminarHabitacion(@Param("id") Long id);
}
```

Escenarios Prueba

1. Mismos PK

Tenemos este registro ya en la tabla reservaciones:

ID	FECHALLEGADA	FECHASALIDA	CANTIDADPERSONAS	HOTEL_ID	PLANCONSUMO_ID	TIPOHABITACION_ID	TITULAR_ID	HABITACIOI
306	30-JAN-24	14-FEB-24	2	1	1	1	100040935	

Y si intentamos agregar un registro con demás valores diferentes pero mismo PK sucede esto:

```
SQL> INSERT INTO reservaciones (ID, FECHALLEGADA, FECHASALIDA, CANTIDADPERSONAS, HOTEL_ID, PLANCONSUMO_ID, TIPOHABITACION_ID,
TITULAR_ID, HABITACION_ID)
  2  VALUES (306, TO_DATE('12-01-2024', 'DD-MM-YYYY'), TO_DATE('10-02-2024', 'DD-MM-YYYY'), 2, 1, 1, 1, 100040935, 266);

INSERT INTO reservaciones (ID, FECHALLEGADA, FECHASALIDA, CANTIDADPERSONAS,
HOTEL_ID, PLANCONSUMO_ID, TIPOHABITACION_ID, TITULAR_ID, HABITACION_ID)
*

ERROR at line 3:
ORA-00001: unique constraint (ISIS2304B08202320.SYS_C001125249) violated
```

2. **FK valida y no valida**

Por ejemplo, para la tabla estadías tiene asociado el id de una cuenta consumo, si el id de la cuenta consumo existe me deja crear el registro sino no deja

```
SQL> INSERT INTO estadias (ID, RESERVA_ID, CUENTACONSUMO_ID, PAZYSALVO, CHECKIN, CHECKOUT, CHECKIN_REALIZADO,
CHECKOUT_REALIZADO)
  2  VALUES (1000, 99999999, 601, 1, TO_DATE('21-12-2022', 'DD-MM-YYYY'), TO_DATE('24-12-2022', 'DD-MM-YYYY'), 0, 0);

INSERT INTO estadias (ID, RESERVA_ID, CUENTACONSUMO_ID, PAZYSALVO, CHECKIN,
CHECKOUT, CHECKIN_REALIZADO, CHECKOUT_REALIZADO)
*

ERROR at line 3:
ORA-02291: integrity constraint (ISIS2304B08202320.FK_ESTADIA_CUENTACONSUMO)
violated - parent key not found
```

3. **Tuplas que violan restricciones**

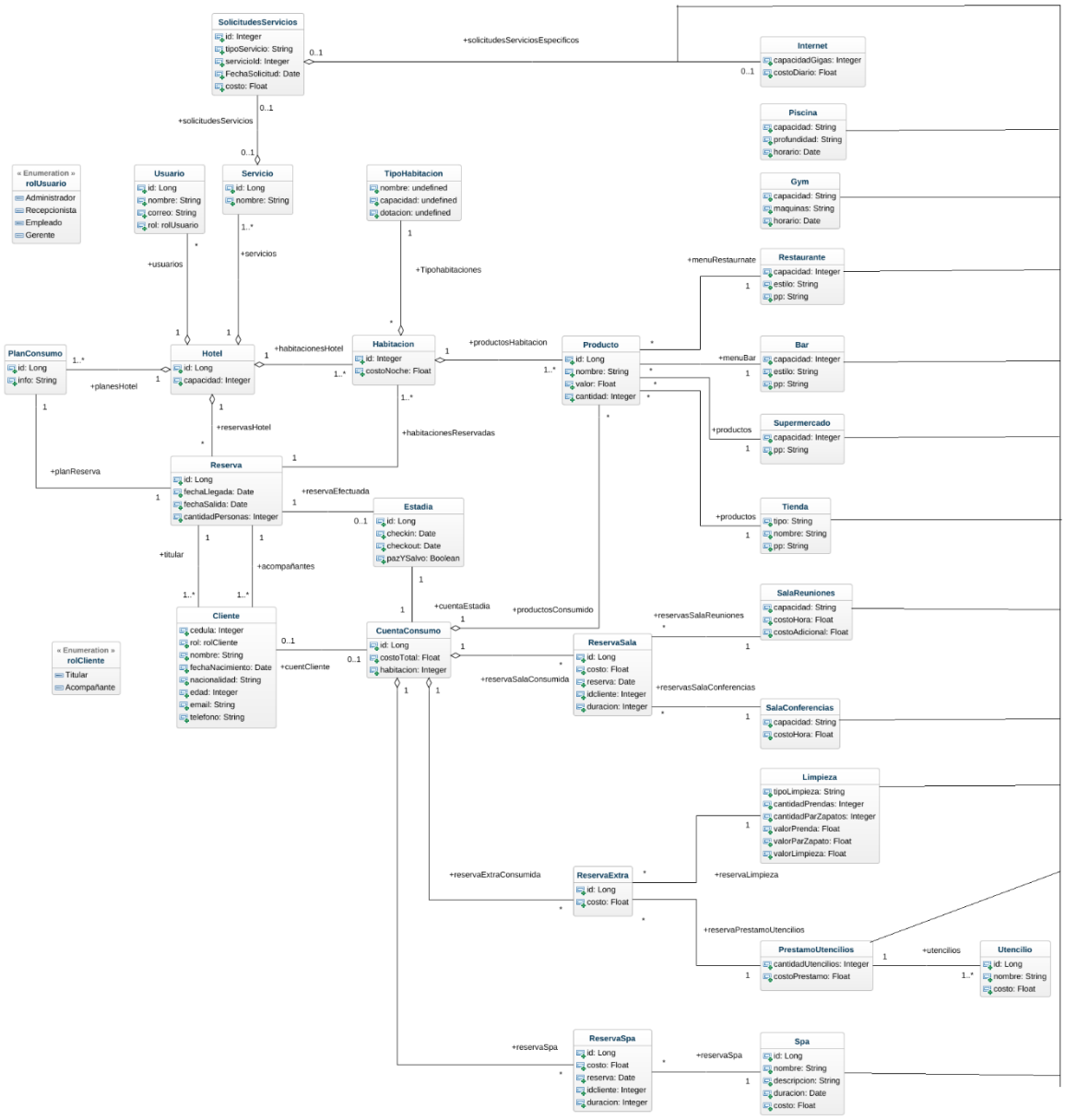
   Tuplas que violan restricciones tenemos que para en este caso todos los numero tienen que se int y cuando los cambio a cadenas no funciona

```
SQL> INSERT INTO reservaciones (ID, FECHALLEGADA, FECHASALIDA, CANTIDADPERSONAS, HOTEL_ID, PLANCONSUMO_ID, TIPOHABITACION_ID,
TITULAR_ID, HABITACION_ID)
  2  VALUES ("hola", TO_DATE('12-01-2024', 'DD-MM-YYYY'), TO_DATE('10-02-2024', 'DD-MM-YYYY'), "2", "1", 1, 1, 100040935, "266");

VALUES ("hola", TO_DATE('12-01-2024', 'DD-MM-YYYY'), TO_DATE('10-02-2024',
'DD-MM-YYYY'), "2", "1", 1, 1, 100040935, "266")
                                *
ERROR at line 4:
ORA-00984: column not allowed here
```

iteración 2:

Análisis BD y UML

Analizando nuestra BD y UML encontramos ligeros cambios con respecto a estos en base a los requerimientos funcionales de esta iteración, los cambios que tenemos que realizar es agregar a las tablas de reservaspa y reservasalon columnas de id titular y duración, además otro cambio que encontramos fue la clase cliente y cuentaconsumo se asociaran directamente en lugar de como lo teníamos anteriormente que era una cuentaconsumo conoce estadía la estadía conoce a la reserva y la reserva conoce al cliente, esto era ineficiente y para las consultas sql esto será de mucha importancia para optimizar las mismas, además decidimos no trabajar los servicios como herencias ya que vimos la necesidad de crear una nueva clase que se llama solicitudes_Servicios esto con la facilidad también de algunos requerimientos



Requerimientos funcionales, planes, escenarios de ejecución y índices

Para la iteración 2 todos los requerimientos funcionales fueron logrados en base a nuestra nuestra base de datos entonces a continuación se presentará cada requerimiento el plan propuesto, si tiene índices o no, la justificación de los índices y el plan Oracle.

REQ1:

```
--REQ 1

SELECT HABITACION, SUM(COSTOTOTAL) AS DINERO_RECOLECTADO
FROM cuentasconsumo
WHERE FECHADELCONSUMO BETWEEN ADD_MONTHS(SYSDATE, -12) AND SYSDATE
GROUP BY HABITACION;
```

Plan propuesto por Oracle:

```
Plan hash value: 1587136782

-----------------------------------------------------------------------------------------
| Id  | Operation           | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                |   279 |  4743 |   119   (1)| 00:00:01 |
|   1 |  HASH GROUP BY      |                |   279 |  4743 |   119   (1)| 00:00:01 |
|*  2 |   FILTER            |                |       |       |            |          |
|*  3 |    TABLE ACCESS FULL| CUENTASCONSUMO |   399 |  6783 |   118   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

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

   2 - filter(SYSDATE@!>=ADD_MONTHS(SYSDATE@!,(-12)))
   3 - filter("FECHADELCONSUMO">=ADD_MONTHS(SYSDATE@!,(-12)) AND
              "FECHADELCONSUMO"<=SYSDATE@!)
```

Explicación del plan propuesto por oracle:

- Plan hash value: Este es un identificador único para el plan de ejecución generado por el optimizador. Puedes usar este valor para comparar este plan con otros y ver si algún cambio en la consulta o en la base de datos (como la adición de índices) afecta el plan de ejecución.

- SELECT STATEMENT: Representa la operación de alto nivel, la ejecución de la sentencia SELECT completa. No está asociado con un costo directamente porque es la suma de todas las operaciones anidadas.

- HASH GROUP BY: Esta operación está realizando una agrupación de los datos basada en una o más columnas especificadas en la cláusula GROUP BY de la consulta. Se usa un algoritmo de hash para realizar esta operación de manera eficiente en memoria.

- FILTER: El filtro es una operación que restringe las filas a aquellas que cumplen una condición booleana específica.

- TABLE ACCESS FULL: Esta es una operación de acceso a los datos en la que la base de datos lee todas las filas de la tabla `CUENTASCONSUMO`, y no solo un subconjunto de ellas. Esto generalmente ocurre cuando no hay índices que puedan ser utilizados para filtrar los datos, o cuando el optimizador determina que un escaneo completo es más eficiente.

La sección Predicate Information proporciona detalles sobre las condiciones aplicadas en las operaciones de filtro:

- Operation Id 2: Esta condición de filtro está verificando si la fecha actual (`SYSDATE`) es mayor o igual que la fecha de hace 12 meses. Este filtro no está asociado directamente con una operación de acceso a la tabla; más bien, podría estar limitando los resultados después de que se hayan agrupado.

- Operation Id 3: Este filtro está aplicado directamente en la operación de acceso a la tabla `CUENTASCONSUMO` y restringe las filas a aquellas cuya columna `FECHADELCONSUMO` cae dentro de los últimos 12 meses desde la fecha actual.

Plan propuesto por nosotros:

Es importante destacar que el acceso completo a la tabla sugiere que no se están utilizando índices para mejorar el rendimiento de la consulta. En bases de datos grandes y con consultas frecuentes por rangos de fechas, un índice en la columna `FECHADELCONSUMO` podría mejorar significativamente el rendimiento.

```
Plan hash value: 2710305915

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                     |   279 |  4743 |    31   (4)| 00:00:01 |
|   1 |  HASH GROUP BY                      |                     |   279 |  4743 |    31   (4)| 00:00:01 |
|*  2 |   FILTER                            |                     |       |       |            |          |
|   3 |    TABLE ACCESS BY INDEX ROWID BATCHED| CUENTASCONSUMO      |   399 |  6783 |    30   (0)| 00:00:01 |
|*  4 |     INDEX RANGE SCAN                | IDX_FECHADELCONSUMO |   399 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------

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

   2 - filter(SYSDATE@!>=ADD_MONTHS(SYSDATE@!,(-12)))
   4 - access("FECHADELCONSUMO">=ADD_MONTHS(SYSDATE@!,(-12)) AND "FECHADELCONSUMO"<=SYSDATE@!)
```

Comparación de los planes:

El segundo plan de ejecución muestra un acceso más eficiente a los datos debido al uso del índice IDX_FECHADELCONSUMO. Aquí está la comparación entre los dos planes:

Primer Plan:

- No utiliza un índice; en su lugar, realiza un acceso completo a la tabla (TABLE ACCESS FULL) para la tabla CUENTASCONSUMO.

- El costo total estimado de la consulta es 119 unidades de costo con un componente de CPU muy bajo (1%).

- El acceso completo a la tabla significa que Oracle debe leer todas las filas de la tabla y luego filtrarlas para cumplir con las condiciones de la consulta.

Segundo Plan (con índice):

- Utiliza INDEX RANGE SCAN, lo que indica que Oracle puede buscar de manera eficiente el rango de filas que coinciden con las condiciones del filtro utilizando el índice IDX_FECHADELCONSUMO.

- La operación TABLE ACCESS BY INDEX ROWID BATCHED muestra que Oracle primero usa el índice para encontrar los ROWIDs de las filas relevantes y luego accede a estas filas directamente en la tabla.

- El costo total estimado de esta consulta es significativamente menor, con solo 31 unidades de costo. Además, el porcentaje de CPU también es bajo.

El uso del índice reduce el costo de la consulta y, presumiblemente, el tiempo de ejecución, ya que la base de datos puede localizar las filas de interés mucho más rápidamente que al leer toda la tabla. Esto es particularmente efectivo para tablas grandes donde la lectura de toda la tabla sería costosa en términos de recursos y tiempo.

REQ2

```
--REQ 2

SELECT
    SERVICIO_TYPE AS NOMBRESERVICIO,
    COUNT(*) AS VECESSOLICITADO
FROM
    solicitudes_servicios
WHERE
    FECHA_SOLICITUD BETWEEN :fechaInicio AND :fechaFin
GROUP BY
    SERVICIO_TYPE
ORDER BY
    VECESSOLICITADO DESC
FETCH FIRST 20 ROWS ONLY;
```

PLAN ORACLE

```
Plan hash value: 2446289128

-------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                     |    20 |  1320 |     7  (43)| 00:00:01 |
|   1 |  SORT ORDER BY           |                     |    20 |  1320 |     7  (43)| 00:00:01 |
|*  2 |   VIEW                   |                     |    20 |  1320 |     6  (34)| 00:00:01 |
|*  3 |    WINDOW SORT PUSHED RANK|                     |     2 |    36 |     6  (34)| 00:00:01 |
|   4 |     HASH GROUP BY        |                     |     2 |    36 |     6  (34)| 00:00:01 |
|*  5 |      FILTER              |                     |       |       |            |          |
|*  6 |       TABLE ACCESS FULL  | SOLICITUDES_SERVICIOS |     2 |    36 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------

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

   2 - filter("from$_subquery$_002"."rowlimit_$$_rownumber"<=20)
   3 - filter(ROW_NUMBER() OVER ( ORDER BY COUNT(*) DESC )<=20)
   5 - filter(TO_DATE(:FECHAFIN)>=TO_DATE(:FECHAINICIO))
   6 - filter("FECHA_SOLICITUD">=:FECHAINICIO AND "FECHA_SOLICITUD"<=:FECHAFIN)
```

1. SELECT STATEMENT: La capa más externa del plan de ejecución que representa la ejecución de la consulta completa.

2. SORT ORDER BY: Oracle necesita ordenar el resultado de la consulta según algún criterio, que es necesario para las operaciones de clasificación y límite de filas.

3. VIEW: Indica que la consulta está utilizando una vista o una subconsulta que Oracle ha optimizado como una vista.

4. WINDOW SORT PUSHED RANK: Esta es una operación de clasificación para una función de ventana que probablemente esté utilizando ROW_NUMBER para asignar un rango a cada fila. El "pushed rank" significa que Oracle intenta optimizar el rendimiento empujando el cálculo de rango hacia abajo en el plan de ejecución tanto como sea posible.

5. HASH GROUP BY: Oracle está agrupando las filas por alguna columna (o columnas) para realizar un cálculo agregado, en este caso, el conteo de filas.

6. FILTER: Hay un filtro aplicado en el paso de la operación de grupo, pero sin más detalles, es difícil decir qué está filtrando exactamente.

7. TABLE ACCESS FULL: Oracle está realizando un acceso completo a la tabla SOLICITUDES_SERVICIOS, lo que implica que está leyendo todas las filas de la tabla porque no hay índices que puedan ser utilizados o los índices disponibles no son útiles para la consulta actual.

PLAN PROPUESTO POR NOSTOROS

```
Plan hash value: 2496378305

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name                 | Rows  | Bytes | Cost (%CPU)| Time
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                      |    20 |  1320 |     5  (60)| 00:00:(
|   1 |  SORT ORDER BY                         |                      |    20 |  1320 |     5  (60)| 00:00:(
|*  2 |   VIEW                                 |                      |    20 |  1320 |     4  (50)| 00:00:(
|*  3 |    WINDOW SORT PUSHED RANK             |                      |     2 |    36 |     4  (50)| 00:00:(
|   4 |     HASH GROUP BY                      |                      |     2 |    36 |     4  (50)| 00:00:(
|*  5 |      FILTER                            |                      |       |       |            |
|   6 |       TABLE ACCESS BY INDEX ROWID BATCHED| SOLICITUDES_SERVICIOS |     2 |    36 |     2   (0)| 00:00:(
|*  7 |        INDEX RANGE SCAN                | IDX_FECHA_SOLICITUD  |     2 |       |     1   (0)| 00:00:(
-----------------------------------------------------------------------------------------------------------------------

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

   2 - filter("from$_subquery$_002"."rowlimit_$$_rownumber"<=20)
   3 - filter(ROW_NUMBER() OVER ( ORDER BY COUNT(*) DESC )<=20)
   5 - filter(TO_DATE(:FECHAFIN)>=TO_DATE(:FECHAINICIO))
   7 - access("FECHA_SOLICITUD">=:FECHAINICIO AND "FECHA_SOLICITUD"<=:FECHAFIN)
```

Es funcional por varias razones:

```
CREATE INDEX idx_fecha_solicitud ON solicitudes_servicios(FECHA_SOLICITUD);
```

1. Filtrado de Rangos de Fechas: La consulta utiliza un rango de fechas para filtrar los resultados (`"FECHA_SOLICITUD">=:FECHAINICIO AND "FECHA_SOLICITUD"<=:FECHAFIN`). Un índice en `FECHA_SOLICITUD` permite al motor de la base de datos localizar rápidamente las filas que caen dentro de este rango sin tener que escanear toda la tabla. Esto es especialmente útil si la tabla es grande y solo un subconjunto de las filas cae dentro del rango de fechas especificado.

2. Eficiencia en el Acceso a Datos: Cuando se realiza una consulta que filtra por una columna indexada, el motor de la base de datos puede utilizar el índice para acceder directamente a las filas relevantes en lugar de realizar una lectura completa de la tabla (FULL TABLE ACCESS), lo que es mucho más lento.

3. Mejora en las Operaciones de Ordenamiento y Agrupamiento: Si bien el índice propuesto no está directamente relacionado con las operaciones de GROUP BY y ORDER BY mencionadas en el plan de ejecución, tener un índice en las columnas que son comúnmente filtradas o agrupadas puede mejorar indirectamente el rendimiento de estas operaciones al reducir el conjunto de datos sobre el cual se tienen que realizar.

4. Optimización de las Funciones de Ventana: En la consulta se utiliza `ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC)`, que es una función de ventana para asignar un rango. Aunque el índice no mejorará directamente el rendimiento de esta función de ventana, al reducir el número de filas mediante el filtrado eficiente, hay menos datos sobre los que computar la función, lo que puede resultar en una mejora de rendimiento.

5. Uso en Consultas Futuras: Si bien la consulta actual puede no ser la única que se ejecuta contra esta tabla, un índice en `FECHA_SOLICITUD` puede ser beneficioso para otras consultas que también utilicen esta columna para filtrado o para ordenar los resultados.

COMPARACION DE PLANES

Al comparar el nuevo plan de ejecución que utiliza el índice `IDX_FECHA_SOLICITUD` con el plan anterior que no lo utilizaba, se pueden observar las siguientes diferencias:

1. Costo Total de la Consulta:

- El plan anterior tenía un costo total de 7 unidades de costo.
- El nuevo plan tiene un costo total de 5 unidades de costo.

El uso del índice ha reducido el costo total de la consulta, lo que sugiere que el motor de base de datos espera que la consulta sea más eficiente y requiera menos recursos para completarse.

2. Acceso a la Tabla:

- El plan anterior realizaba un acceso completo a la tabla (TABLE ACCESS FULL).

- El nuevo plan realiza un TABLE ACCESS BY INDEX ROWID BATCHED, indicando que utiliza el índice para encontrar las filas pertinentes y luego accede a la tabla directamente a través de los ROWIDs encontrados en el índice.

3. Uso del Índice:

- El plan anterior no mostraba ninguna operación que hiciera uso de un índice.

- El nuevo plan realiza un INDEX RANGE SCAN en `IDX_FECHA_SOLICITUD`, lo que significa que está utilizando el índice para filtrar eficientemente las filas basadas en el rango de fechas especificado en la consulta.

4. Predicados de Filtro:

- Ambos planes utilizan filtros similares, pero en el nuevo plan, el predicado asociado con el índice (access("FECHA_SOLICITUD">=:FECHAINICIO AND "FECHA_SOLICITUD"<=:FECHAFIN)) muestra que está efectivamente haciendo uso del índice para limitar las filas que necesitan ser consideradas.

En resumen, el nuevo plan es más eficiente debido al uso del índice, permitiendo al motor de base de datos reducir significativamente el número de filas que necesita examinar y, por tanto, reducir el costo de la consulta. Esto es especialmente beneficioso en tablas grandes donde la lectura completa de la tabla puede ser muy costosa en términos de tiempo y uso de recursos. El índice proporciona un acceso directo y rápido a las filas relevantes.

REQ3

```
---REQ 3
SELECT * FROM reservaciones;

SELECT HABITACION_ID,
|   |   (COUNT(*) * 100.0 / (SELECT COUNT(*) FROM reservaciones WHERE FECHALLEGADA BETWEEN AI
FROM reservaciones
WHERE FECHALLEGADA BETWEEN ADD_MONTHS(SYSDATE, -12) AND SYSDATE
GROUP BY HABITACION_ID;
```

PLAN ORACLE

```
Plan hash value: 3434362556

-------------------------------------------------------------------------------
| Id  | Operation         | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |               |   499 | 21457 |   118   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| RESERVACIONES |   499 | 21457 |   118   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

El plan de ejecución indica lo siguiente: Id 0: Es el nodo raíz del plan, que indica que se realizará una declaración SELECT sin ninguna operación de filtrado o indexación adicional. Id 1: Es un acceso completo a la tabla (TABLE ACCESS FULL), lo que significa que la base de datos leerá todas las filas de la tabla reservaciones para encontrar las que cumplen con el criterio de la cláusula WHERE. El plan también muestra: Rows: El optimizador estima que se leerán 499 filas durante la ejecución de la consulta. Bytes: El optimizador estima que el total de bytes leídos será de 21457. Cost (%CPU): El costo estimado para ejecutar esta operación es de 118 unidades de costo, con un porcentaje del CPU esperado del 0% (lo que sugiere que el costo está más relacionado con la lectura de disco que con el procesamiento del CPU). Time: El tiempo estimado para la operación es de 1 segundo.

PLAN PROPUESTO POR NOSTROS:

Para mejorar la eficiencia de la consulta que estás ejecutando, podrías considerar crear un índice sobre la columna `FECHALLEGADA` de la tabla `reservaciones`. Dado que la consulta filtra las reservaciones basándose en un rango de fechas que abarca los últimos 12 meses, un índice en esa columna podría ayudar a reducir el número de filas que deben ser examinadas y, por lo tanto, acelerar la operación de filtrado.

```
CREATE INDEX idx_fechallegada ON reservaciones(FECHALLEGADA);
```

Con este índice, el optimizador de consultas de Oracle tiene una mayor probabilidad de utilizar un `INDEX RANGE SCAN` para acceder rápidamente a las filas que cumplen con la condición de la cláusula `WHERE`. Esto sería especialmente útil si el rango de fechas no incluye la mayoría de las filas de la tabla (es decir, si no es una operación que abarque toda la tabla).

COMPARACION DE LOS PLANES:

Dada nuestra sentencia sql y la creación del índice el plan sigue siendo el mismo por lo que la creación del índice no era necesaria esto se debe a que el plan de por si ya era lo suficientemente eficiente

REQ4

```
--REQ 4

SELECT * FROM solicitudes_servicios
WHERE
    (COSTO BETWEEN :precio_min AND :precio_max OR (:precio_min IS NULL AND :precio_max IS NU
AND
    (FECHA_SOLICITUD BETWEEN TO_DATE(:fecha_inicio, 'DD-MON-YY') AND TO_DATE(:fecha_fin, 'DI
AND
    (SERVICIO_TYPE LIKE '%' || :nombre_servicio || '%' OR :nombre_servicio IS NULL);
```

PLAN ORACLE

```
Plan hash value: 370729998

-----------------------------------------------------------------------------------------
| Id  | Operation         | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                     |     1 |    28 |     4   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| SOLICITUDES_SERVICIOS |     1 |    28 |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

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

   1 - filter((:PRECIO_MIN IS NULL AND :PRECIO_MAX IS NULL OR
              "COSTO">=TO_NUMBER(:PRECIO_MIN) AND "COSTO"<=TO_NUMBER(:PRECIO_MAX)) AND
              (:NOMBRE_SERVICIO IS NULL OR "SERVICIO_TYPE" LIKE '%'||:NOMBRE_SERVICIO||'%') AND
              (:FECHA_INICIO IS NULL AND :FECHA_FIN IS NULL OR
              "FECHA_SOLICITUD">=TO_DATE(:FECHA_INICIO,'DD-MON-YY') AND
              "FECHA_SOLICITUD"<=TO_DATE(:FECHA_FIN,'DD-MON-YY')))
```

1. Operation: `TABLE ACCESS FULL` sobre `SOLICITUDES_SERVICIOS`

- Esto significa que Oracle ha decidido realizar un escaneo completo de la tabla `SOLICITUDES_SERVICIOS`. No está utilizando ningún índice para restringir la búsqueda, lo que generalmente ocurre cuando se espera que la consulta acceda a una gran parte de la tabla o cuando no existe un índice apropiado.

- La operación es la única en el plan, por lo que es la acción principal que se está llevando a cabo.

2. Rows: `1`

- Oracle estima que la consulta va a recuperar aproximadamente una fila. Esta estimación se basa en las estadísticas de la tabla, que deben estar actualizadas para que esta estimación sea precisa.

3. Bytes: `28`

- Se espera que cada fila recuperada tenga un tamaño de 28 bytes.

4. Cost (%CPU): `4 (0)`

- El coste estimado para realizar esta operación es de 4 unidades de coste de Oracle. Cuanto menor sea el número, menos recursos se espera que consuma la consulta. El porcentaje entre paréntesis indica la cantidad de ese coste que se espera que sea uso de CPU, que en este caso es 0%, lo que sugiere que la operación estará limitada principalmente por la lectura de la entrada/salida (I/O).

5. Time: `00:00:01`

- Se estima que la operación tomará aproximadamente un segundo. Esto es una estimación y el tiempo real puede variar.

La sección Predicate Information proporciona detalles sobre las condiciones de filtrado aplicadas en la operación:

- El filtro en la operación `1` es complejo e involucra varias condiciones que pueden contener valores `NULL`. Parece ser una consulta que permite a los usuarios especificar un rango de

precios, un nombre de servicio y un rango de fechas, todos los cuales son opcionales. Si no se proporcionan (es decir, son `NULL`), no se aplican como parte del filtro.

La estructura del filtro sugiere que hay una serie de parámetros que el usuario puede pasar a la consulta, y estos parámetros pueden incluir:

- `:PRECIO_MIN` y `:PRECIO_MAX` para filtrar los servicios por un rango de costos.

- `:NOMBRE_SERVICIO` para buscar servicios que contengan cierto texto en su tipo.

- `:FECHA_INICIO` y `:FECHA_FIN` para filtrar servicios basados en la fecha de solicitud.

Para cada uno de estos parámetros, si no se proporciona un valor (es decir, el parámetro es `NULL`), la consulta no utilizará ese criterio específico como filtro. Por ejemplo, si `:PRECIO_MIN` y `:PRECIO_MAX` son `NULL`, la consulta no filtrará por el costo.

PLAN PROPUESTO POR NOSTROS:

Para mejorar el rendimiento de las consultas que utilizan las columnas `SERVICIO_TYPE`, `FECHA_SOLICITUD` y `COSTO`, podríamos considerar la creación de índices en estas columnas. Aquí tienes los comandos SQL para crear índices en esas columnas en Oracle:

Estos índices permitirán búsquedas más rápidas en las columnas respectivas, especialmente si las consultas frecuentemente filtran o ordenan por estas columnas. No obstante, es importante tener en cuenta lo siguiente:

- Cada índice adicional consume más espacio en disco y puede hacer que las operaciones de inserción, actualización o eliminación en la tabla sean más lentas, ya que los índices deben ser actualizados.

- La utilidad de un índice depende de la selectividad de la columna; es decir, cuánto puede reducir el conjunto de resultados. Si casi todas las filas tienen valores distintos (alta cardinalidad), como podría ser el caso de `COSTO`, un índice puede no ser tan efectivo.

- El índice en `FECHA_SOLICITUD` probablemente será muy útil para consultas que filtran por rangos de fechas.

- El índice en `SERVICIO_TYPE` sería más útil si las consultas buscan coincidencias exactas o utilizan condiciones `LIKE` con patrones no liderados por comodines

COMPARACION DE LOS PLANES:

Similar al anterior requerimiento tuvimos el mismo plan pese a crear los índices esto indica que Oracle estima que la cantidad de filas que satisfarán las condiciones del filtro es muy baja (solo 1 fila), por lo que considera que un escaneo completo es más eficiente que usar un índice. Los filtros aplicados en la consulta son complejos y podrían incluir operadores que no son sencillos de optimizar con índices (como el uso de LIKE con comodines en ambos extremos). Las estadísticas de la tabla no están actualizadas, lo que podría llevar al optimizador a tomar decisiones subóptimas.

REQ 5

```
--REQ 5

SELECT c.CLIENTE, cl.NOMBRE, c.FECHADELCONSUMO, c.COSTOTOTAL
FROM cuentasconsumo c
JOIN Clientes cl ON c.CLIENTE = cl.CEDULA
WHERE c.CLIENTE = :cedulaUsuario
AND c.FECHADELCONSUMO BETWEEN TO_DATE(:fechaInicio, 'DD-MON-YY') AND TO_DATE(:fechaFin, 'DD
ORDER BY c.FECHADELCONSUMO;
```

PLAN PROPUESTO POR NOSOTROS

Para este requerimiento 5 ya habíamos creado el plan entonces no alcanzamos a compararlo con el de oracle

```
Plan hash value: 1247948108

-----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                     |     1 |    43 |     4   (0)| 00:00:01 |
|*  1 |  FILTER                      |                     |       |       |            |          |
|   2 |   NESTED LOOPS               |                     |     1 |    43 |     4   (0)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID| CLIENTES            |     1 |    23 |     2   (0)| 00:00:01 |
|*  4 |     INDEX UNIQUE SCAN        | SYS_C001125226      |     1 |       |     1   (0)| 00:00:01 |
|*  5 |    TABLE ACCESS BY INDEX ROWID| CUENTASCONSUMO      |     1 |    20 |     2   (0)| 00:00:01 |
|*  6 |     INDEX RANGE SCAN         | IDX_FECHADELCONSUMO |     2 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   1 - filter(TO_DATE(:FECHAFIN,'DD-MON-YY')>=TO_DATE(:FECHAINICIO,'DD-MON-YY'))
   4 - access("CL"."CEDULA"=TO_NUMBER(:CEDULAUSUARIO))
   5 - filter("C"."CLIENTE"=TO_NUMBER(:CEDULAUSUARIO))
   6 - access("C"."FECHADELCONSUMO">=TO_DATE(:FECHAINICIO,'DD-MON-YY') AND
              "C"."FECHADELCONSUMO"<=TO_DATE(:FECHAFIN,'DD-MON-YY'))
```

El plan de ejecución proporcionado indica que la consulta está optimizada para el uso de índices, lo que hace que sea eficiente por varias razones:

1. Acceso por Índice Único (Operación 4): La consulta utiliza un "INDEX UNIQUE SCAN" en la tabla `CLIENTES`. Este tipo de acceso es uno de los más rápidos, ya que el índice único garantiza que solo habrá un registro coincidente (o ninguno), lo que minimiza el número de lecturas de disco necesarias para localizar la fila.

2. Acceso por Índice de Rango (Operación 6): La consulta también realiza un "INDEX RANGE SCAN" sobre `IDX_FECHADELCONSUMO`. Esto significa que la consulta puede utilizar un rango de valores para buscar eficientemente dentro del índice, lo que es mucho más rápido que escanear la tabla completa, especialmente si la tabla es grande.

3. Bucle Anidado (Operaciones 2, 5): El plan utiliza "NESTED LOOPS", lo que indica que para cada fila de la tabla `CLIENTES`, busca las filas correspondientes en `CUENTASCONSUMO`. Esto es eficiente cuando el primer conjunto de filas es pequeño, y el acceso a la segunda tabla se optimiza con un índice.

4. Filtrado Temprano (Operación 1): El plan muestra que el filtro de las fechas se realiza al principio (operación 1). Esto reduce la cantidad de datos que necesitan ser procesados en las operaciones posteriores.

5. Costo Bajo (Columna Cost): El costo total de la consulta es bajo (4 unidades de costo), lo que sugiere que la base de datos estima que la consulta se completará rápidamente. El porcentaje del CPU es del 0%, lo que indica que la operación es mayormente de IO y que el uso del índice minimiza la necesidad de procesamiento.

En resumen, el uso de índices reduce significativamente el tiempo de acceso a los datos necesarios y minimiza la cantidad de trabajo que la base de datos tiene que realizar para ejecutar la consulta. Esto es especialmente útil para grandes conjuntos de datos donde las búsquedas de tabla completa serían ineficientes. Este plan demuestra que la consulta está bien optimizada para el rendimiento y es un buen ejemplo de cómo el diseño apropiado de índices puede mejorar significativamente la eficiencia de las operaciones de base de datos.

REQ6

```sql
--REQ 6

SELECT FECHALLEGADA, COUNT(HABITACION_ID) AS NumeroHabitacionesOcupadas
FROM reservaciones
GROUP BY FECHALLEGADA
ORDER BY NumeroHabitacionesOcupadas DESC;

SELECT FECHALLEGADA, COUNT(HABITACION_ID) AS NumeroHabitacionesOcupadas
FROM reservaciones
GROUP BY FECHALLEGADA
ORDER BY NumeroHabitacionesOcupadas ASC;

SELECT FECHADELCONSUMO, SUM(COSTOTOTAL) AS IngresosTotales
FROM cuentasconsumo
GROUP BY FECHADELCONSUMO
ORDER BY IngresosTotales DESC;
```

PLAN ORACLE

```
Plan hash value: 2888089155

-------------------------------------------------------------------------------------
| Id  | Operation           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |               |   499 |  5988 |   120   (2)| 00:00:01 |
|   1 |  SORT ORDER BY      |               |   499 |  5988 |   120   (2)| 00:00:01 |
|   2 |   HASH GROUP BY     |               |   499 |  5988 |   120   (2)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| RESERVACIONES |   499 |  5988 |   118   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

1. Acceso a la Tabla Completa (Operación 3):

- La consulta realiza un "TABLE ACCESS FULL" en la tabla `RESERVACIONES`. Esto significa que no hay índices que la consulta pueda utilizar para filtrar o acceder a las filas, o que la base de datos ha determinado que sería más eficiente escanear toda la tabla. Un escaneo completo puede ser adecuado si la tabla es pequeña, si los índices no son útiles para la consulta, o si se necesita una gran proporción de las filas de la tabla.

2. Agrupación por Hash (Operación 2):

- Se aplica un "HASH GROUP BY", que es un método eficiente para agrupar datos cuando se espera un número considerable de grupos. La operación utiliza una estructura de datos de hash para agrupar rápidamente las filas que tienen las mismas claves de grupo.

3. Ordenación (Operación 1):

- La consulta requiere que los resultados estén ordenados, como lo indica la operación "SORT ORDER BY". Este paso puede ser necesario para los resultados finales según la cláusula ORDER BY de la consulta SQL.

4. Estimación de Filas y Costo:

- La base de datos estima que se recuperarán 499 filas, lo que sugiere que espera que el resultado del grupo por tenga esa cantidad de grupos únicos.

- El costo total de la consulta es de 120 unidades, con un 2% de CPU, indicando que la consulta es más intensiva en IO (entrada/salida) que en procesamiento. El tiempo estimado para la ejecución es de un segundo.

PLAN PROPUESTO POR NOSOTROS

El índice en la columna `FECHALLEGADA` de la tabla `reservaciones` optimiza las dos primeras sentencias SQL por las siguientes razones:

```
CREATE INDEX idx_fechallegada ON reservaciones(FECHALLEGADA);
```

1. Optimización de la Agrupación (`GROUP BY`):

Las consultas realizan una operación de agrupación por la columna `FECHALLEGADA`. Un índice en esta columna puede acelerar la agrupación porque la base de datos puede utilizar el índice para acceder rápidamente a los datos en orden. Esto es especialmente útil cuando hay un gran número de filas. En lugar de leer la tabla entera, la base de datos puede leer el índice mucho más compacto para obtener los valores únicos necesarios para la agrupación.

2. Optimización de la Ordenación (`ORDER BY`):

Después de agrupar los registros por `FECHALLEGADA`, las consultas ordenan los resultados por `NumeroHabitacionesOcupadas`. Si bien el índice en `FECHALLEGADA`

no ayuda directamente a ordenar por `NumeroHabitacionesOcupadas`, la eficiencia en la fase de agrupación conduce a un conjunto de resultados más pequeño que luego se puede ordenar más rápidamente.

3. Reducción de Costos de E/S:

El uso de un índice reduce la cantidad de E/S de disco (lectura de datos) necesaria, ya que leer un índice es generalmente más rápido que leer una tabla completa. Los índices están estructurados de tal manera (usualmente en estructuras de árbol B) que hacen que las operaciones de búsqueda y acceso a los datos sean más eficientes.

4. Mejora en la Velocidad de Acceso a Datos:

Cuando se ejecuta la agrupación, si la base de datos puede confiar en un índice para acceder a los datos preordenados, puede mejorar significativamente la velocidad con la que los datos se recuperan y se procesan para la agrupación.

COMPARACION DE PLANES:

Sin Índices (Plan Anterior):

- Escanear Tabla Completa: Sin un índice, la base de datos debe realizar una operación de escaneo completo de la tabla. Esto significa que cada fila de la tabla debe ser leída y procesada para determinar si cumple con los criterios de la consulta, lo que puede ser muy ineficiente para tablas grandes.

- Mayor Uso de CPU y E/S: El procesamiento de cada fila individualmente requiere más ciclos de CPU y operaciones de E/S, ya que los datos no están pre-organizados de ninguna manera que beneficie a la consulta.

- Ordenación en Tiempo de Ejecución: Sin índices, los datos deben ser ordenados en tiempo de ejecución después de ser recuperados, lo que puede ser un proceso intensivo si el conjunto de resultados es grande.

Con Índices (Plan Propuesto):

- Acceso Rápido a Datos: Un índice sobre `FECHALLEGADA` permite a la base de datos saltar rápidamente a las filas relevantes sin tener que leer toda la tabla. Los índices están generalmente organizados como árboles B (o variantes), lo que permite búsquedas rápidas y acceso secuencial eficiente, ideal para operaciones de agrupación y ordenación.

- Reducción de Carga de CPU y E/S: Dado que el índice reduce la necesidad de lecturas de disco completas, hay menos carga en la E/S de disco y en la CPU. Solo se leen los datos necesarios, y el índice ayuda a evitar el procesamiento de datos irrelevantes.

- Optimización de `GROUP BY` y `ORDER BY`: Con el índice, las filas ya están preordenadas por `FECHALLEGADA`, lo que facilita y acelera la agrupación. Además, si los resultados de la agrupación son pequeños o si la base de datos puede usar el índice para obtener resultados ya ordenados, la operación de ordenación puede ser más rápida o incluso innecesaria.

REQ 7

```sql
--REQ 7

WITH DiasEstadia AS (
    SELECT r.TITULAR_ID AS CLIENTE,
           SUM(TO_DATE(e.CHECKOUT, 'DD-MM-YYYY') - TO_DATE(e.CHECKIN, 'DD-MM-YYYY')) AS DIAS
    FROM reservaciones r
    JOIN estadias e ON r.ID = e.RESERVA_ID
    WHERE e.CHECKIN_REALIZADO = 1 AND e.CHECKOUT_REALIZADO = 1
    GROUP BY r.TITULAR_ID
),
Consumo AS (
    SELECT c.CLIENTE,
           SUM(c.COSTOTOTAL) AS TOTALCONSUMO
    FROM cuentasconsumo c
    WHERE TO_DATE(c.FECHADELCONSUMO, 'DD-MM-YYYY') > (SYSDATE - INTERVAL '1' YEAR)
    GROUP BY c.CLIENTE
)
SELECT DISTINCT d.CLIENTE
FROM DiasEstadia d
LEFT JOIN Consumo co ON d.CLIENTE = co.CLIENTE
WHERE d.DIAS >= 14 OR co.TOTALCONSUMO > 15000000;
```

PLAN PROPUESTO POR NOSOTROS

Para este requerimiento 7 ya habíamos creado el plan entonces no alcanzamos a compararlo con el de oracle

```
Plan hash value: 2534823018

-----------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                |     1 |    40 |   125   (3)| 00:00:01 |
|   1 |  HASH UNIQUE                   |                |     1 |    40 |   125   (3)| 00:00:01 |
|*  2 |   FILTER                       |                |       |       |            |          |
|*  3 |    HASH JOIN OUTER             |                |     9 |   360 |   124   (2)| 00:00:01 |
|   4 |     JOIN FILTER CREATE         | :BF0000        |     1 |    20 |     5  (20)| 00:00:01 |
|   5 |      VIEW                      |                |     1 |    20 |     5  (20)| 00:00:01 |
|   6 |       HASH GROUP BY            |                |     1 |    68 |     5  (20)| 00:00:01 |
|   7 |        NESTED LOOPS            |                |     1 |    68 |     4   (0)| 00:00:01 |
|   8 |         NESTED LOOPS           |                |     1 |    68 |     4   (0)| 00:00:01 |
|*  9 |          TABLE ACCESS FULL     | ESTADIAS       |     1 |    57 |     4   (0)| 00:00:01 |
|* 10 |          INDEX UNIQUE SCAN     | SYS_C001125249 |     1 |       |     0   (0)| 00:00:01 |
|  11 |         TABLE ACCESS BY INDEX ROWID| RESERVACIONES  |     1 |    11 |     0   (0)| 00:00:01 |
|  12 |     VIEW                       |                |    20 |   400 |   119   (1)| 00:00:01 |
|  13 |      HASH GROUP BY             |                |    20 |   400 |   119   (1)| 00:00:01 |
|  14 |       JOIN FILTER USE          | :BF0000        |    20 |   400 |   118   (0)| 00:00:01 |
|* 15 |        TABLE ACCESS FULL       | CUENTASCONSUMO |    20 |   400 |   118   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   2 - filter("D"."DIAS">=14 OR "CO"."TOTALCONSUMO">15000000)
   3 - access("D"."CLIENTE"="CO"."CLIENTE"(+))
   9 - filter("E"."CHECKIN_REALIZADO"=1 AND "E"."CHECKOUT_REALIZADO"=1)
  10 - access("R"."ID"="E"."RESERVA_ID")
```

El plan esta optimizado para la consulta específica que se está ejecutando. Estas son las razones:

1. Uso de `HASH JOIN OUTER` (Operación 3): La unión hash es generalmente más rápida que otras formas de unión cuando se trabaja con grandes conjuntos de datos porque construye una tabla hash en memoria para la tabla más pequeña, lo que permite una búsqueda rápida durante la unión con la tabla más grande.

2. `HASH UNIQUE` (Operación 1): Implica que la consulta puede requerir una salida única de los resultados, y el uso de un hash para esta operación es eficiente en términos de velocidad cuando se compara con un ordenamiento y luego un filtro de duplicados.

3. `TABLE ACCESS FULL` con filtro eficiente (Operaciones 9 y 15): Aunque un acceso completo a la tabla puede parecer ineficiente, si la tabla es lo suficientemente pequeña o si el filtro aplicado reduce significativamente el conjunto de datos a procesar, puede ser muy rápido. Además, si no existe un índice apropiado que pueda ayudar con la consulta, un acceso total a la tabla puede ser la mejor opción.

4. `INDEX UNIQUE SCAN` y `TABLE ACCESS BY INDEX ROWID` (Operaciones 10 y 11)**: Estas operaciones indican que la consulta puede aprovechar un índice único para acceder rápidamente a los datos necesarios de la tabla `RESERVACIONES`, lo que es altamente eficiente para recuperar filas específicas.

5. `JOIN FILTER CREATE` y `JOIN FILTER USE` (Operaciones 4 y 14): Esto sugiere que la base de datos está aplicando un filtro de Bloom para optimizar la unión, lo cual es una forma eficiente de reducir la cantidad de datos que se deben procesar en las operaciones de unión.

6. `HASH GROUP BY` (Operaciones 6 y 13): Similar al hash join, el hash group by es eficiente para agrupar datos cuando los conjuntos de datos son grandes.

7. `FILTER` aplicado después de las operaciones de unión (Operación 2): Los filtros se aplican una vez que se han unido las tablas y no antes, lo que significa que la base de datos no desperdicia recursos aplicando filtros a datos que no estarán en el conjunto de resultados final.

8. Predicados de Filtrado Efectivos (Predicate Information): Las condiciones de filtrado parecen estar bien definidas y se aplican de manera que reducen el número de filas procesadas lo más rápido posible.

REQ 8

```
--REQ 8

SELECT NOMBRESERVICIO, VECESSOLICITADO
FROM (
    SELECT SERVICIO_TYPE AS NOMBRESERVICIO, COUNT(*) AS VECESSOLICITADO
    FROM solicitudes_servicios
    WHERE FECHA_SOLICITUD BETWEEN TO_DATE('01-JAN-2023', 'DD-MON-YYYY') AND TO_DATE('31-DEC
    GROUP BY SERVICIO_TYPE
    HAVING COUNT(*) < (3 * 52)
    ORDER BY VECESSOLICITADO ASC
)
WHERE ROWNUM <= 3;
```

PLAN PROPUESTO POR NOSOTROS

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name                 | Rows  | Bytes | Cost (%CPU)| Time
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                      |     1 |    40 |     4  (50)| 00:00:01
|*  1 |  COUNT STOPKEY                         |                      |       |       |            |
|   2 |   VIEW                                 |                      |     1 |    40 |     4  (50)| 00:00:01
|*  3 |    SORT ORDER BY STOPKEY               |                      |     1 |    18 |     4  (50)| 00:00:01
|*  4 |     HASH GROUP BY                      |                      |     1 |    18 |     4  (50)| 00:00:01
|   5 |      TABLE ACCESS BY INDEX ROWID BATCHED| SOLICITUDES_SERVICIOS |   100 |  1800 |     2   (0)| 00:00:01
|*  6 |       INDEX RANGE SCAN                 | IDX_FECHA_SOLICITUD  |   100 |       |     1   (0)| 00:00:01
-----------------------------------------------------------------------------------------------------------

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

   1 - filter(ROWNUM<=3)
   3 - filter(ROWNUM<=3)
   4 - filter(COUNT(*)<156)
   6 - access("FECHA_SOLICITUD">=TO_DATE(' 2023-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "FECHA_SOLICITUD"<=TO_DATE(' 2023-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

Para este requerimiento 8 ya habíamos creado el plan entonces no alcanzamos a compararlo con el de oracle

Pero nuestro plan esta mas optimizado que el de oracle debido a lo siguiente:

1. `SELECT STATEMENT` (Operación 0): Este es el inicio de la ejecución de la consulta.

2. `COUNT STOPKEY` (Operación 1): Oracle utiliza el `STOPKEY` cuando hay una cláusula ROWNUM en la consulta. Esto indica que la base de datos detendrá la búsqueda tan

pronto como encuentre las 3 primeras filas que coinciden con la condición, lo que es muy eficiente en términos de recursos.

3. `VIEW` (Operación 2): Oracle puede estar usando una vista interna para representar el conjunto de resultados de la consulta subyacente.

4. `SORT ORDER BY STOPKEY` (Operación 3): La consulta necesita ordenar los resultados para aplicar el `STOPKEY`. Esta operación se encarga de ordenar y limitar el conjunto de resultados.

5. `HASH GROUP BY` (Operación 4): Se utiliza para agrupar los resultados de la consulta, lo cual es una operación eficiente para el agrupamiento, especialmente cuando no se esperan grandes conjuntos de resultados.

6. `TABLE ACCESS BY INDEX ROWID BATCHED` (Operación 5): Esta operación indica que Oracle está accediendo a la tabla `SOLICITUDES_SERVICIOS` de una manera eficiente, utilizando un índice para encontrar las filas que necesita sin tener que escanear toda la tabla.

7. `INDEX RANGE SCAN` (Operación 6): Aquí es donde se observa el uso del índice `IDX_FECHA_SOLICITUD`. El rango de fechas proporcionado en la condición `WHERE` de la consulta permite que Oracle utilice este índice para encontrar rápidamente las filas relevantes sin realizar un escaneo completo de la tabla. Esto es especialmente eficiente cuando la tabla es grande y solo un subconjunto de filas está dentro del rango de fechas dado.

La información de predicado muestra las condiciones específicas utilizadas en la consulta:

- La operación 1 y 3 usa `ROWNUM<=3` para limitar los resultados a las primeras 3 filas.

- La operación 4 usa `COUNT(*)<156` como parte del criterio de agrupamiento.

- La operación 6 muestra que el índice se usa para buscar filas donde `FECHA_SOLICITUD` esté dentro del año 2023.

En resumen, este plan de ejecución es eficiente debido al uso del índice `IDX_FECHA_SOLICITUD`, que permite un acceso rápido a las filas relevantes basadas en

el rango de fechas. Además, al combinar esto con `STOPKEY`, Oracle puede minimizar la cantidad de datos leídos y procesados, lo que lleva a una ejecución de consulta más rápida y una menor utilización de recursos.

REQ 9

```sql
--REQ 9

SELECT
    CLIENTE,
    SERVICIO,
    COUNT(*) AS NUMERO_DE_VECES
FROM
    cuentasconsumo
WHERE
    SERVICIO = :servicioSeleccionado
    AND FECHADELCONSUMO BETWEEN :fechaInicio AND :fechaFin
GROUP BY
    CLIENTE, SERVICIO
ORDER BY
    NUMERO_DE_VECES DESC;
```

PLAN PROPUESTO POR NOSOTROS

```
Plan hash value: 1402981419

-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                     |     1 |    21 |     4  (50)| 00:00:01 |
|   1 |  SORT ORDER BY                           |                     |     1 |    21 |     4  (50)| 00:00:01 |
|   2 |   HASH GROUP BY                          |                     |     1 |    21 |     4  (50)| 00:00:01 |
|*  3 |    FILTER                                |                     |       |       |            |          |
|*  4 |     TABLE ACCESS BY INDEX ROWID BATCHED| CUENTASCONSUMO      |     1 |    21 |     2   (0)| 00:00:01 |
|*  5 |      INDEX RANGE SCAN                    | IDX_FECHADELCONSUMO |     2 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------

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

   3 - filter(TO_DATE(:FECHAFIN)>=TO_DATE(:FECHAINICIO))
   4 - filter("SERVICIO"=:SERVICIOSELECCIONADO)
   5 - access("FECHADELCONSUMO">=:FECHAINICIO AND "FECHADELCONSUMO"<=:FECHAFIN)
```

Para este requerimiento 9 ya habíamos creado el plan entonces no alcanzamos a compararlo con el de oracle

Pero nuestro plan esta más optimizado que el de oracle debido a lo siguiente:

1. `SELECT STATEMENT` (Operación 0): Esta es la operación principal que inicia la ejecución de la consulta.

2. `SORT ORDER BY` (Operación 1): Esta operación se encarga de ordenar los resultados según algún criterio. En este caso, probablemente se utiliza para ordenar los resultados después de la agrupación.

3. `HASH GROUP BY` (Operación 2): Aquí se realiza la operación de agrupamiento de datos. Se agrupan los datos en función de las columnas especificadas en la consulta.

4. `FILTER` (Operación 3): Esta operación filtra los datos que cumplen ciertos criterios antes de realizar el agrupamiento y la ordenación. La filtración se realiza en base a las condiciones proporcionadas en el predicado.

5. `TABLE ACCESS BY INDEX ROWID BATCHED` (Operación 4): Esta operación indica que Oracle accede a la tabla `CUENTASCONSUMO` utilizando un índice para encontrar las filas relevantes que cumplen con los criterios de filtrado y agrupación. La búsqueda se realiza en lotes (batched), lo que puede mejorar la eficiencia de la consulta.

6. `INDEX RANGE SCAN` (Operación 5): Aquí es donde se utiliza el índice `IDX_FECHADELCONSUMO`. El índice se escanea en función del rango de fechas proporcionado en la condición `WHERE` de la consulta. Esto permite un acceso rápido a las filas que cumplen con el rango de fechas especificado.

El predicado Information muestra las condiciones específicas utilizadas en la consulta:

- La operación 3 filtra los datos en función de la fecha de consumo (`FECHADELCONSUMO`) y el tipo de servicio (`SERVICIO`) seleccionado.

- La operación 4 indica que se está accediendo a la tabla `CUENTASCONSUMO`, y la operación 5 muestra que se está utilizando el índice `IDX_FECHADELCONSUMO` para buscar filas dentro del rango de fechas especificado.

REQ 10

```sql
--REQ 10

SELECT
    c.CLIENTE,
    c.FECHADELCONSUMO,
    c.SERVICIO
FROM
    cuentasconsumo c
WHERE
    NOT EXISTS (
        SELECT 1
        FROM cuentasconsumo cc
        WHERE cc.CLIENTE = c.CLIENTE
        AND cc.SERVICIO = :servicioSeleccionado
        AND cc.FECHADELCONSUMO BETWEEN :fechaInicio AND :fechaFin
    )
GROUP BY
    c.CLIENTE, c.FECHADELCONSUMO, c.SERVICIO
ORDER BY
    c.CLIENTE, c.FECHADELCONSUMO DESC;
```

PLAN PROPUESTO POR NOSOTROS

```
Plan hash value: 784839668

-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |                     |   399 | 16758 |   121   (1)| 00:00:01 |
|   1 |  SORT GROUP BY                       |                     |   399 | 16758 |   121   (1)| 00:00:01 |
|*  2 |   HASH JOIN RIGHT ANTI               |                     |   399 | 16758 |   120   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS BY INDEX ROWID BATCHED| CUENTASCONSUMO      |     1 |    21 |     2   (0)| 00:00:01 |
|*  4 |     INDEX RANGE SCAN                 | IDX_FECHADELCONSUMO |     2 |       |     1   (0)| 00:00:01 |
|   5 |    TABLE ACCESS FULL                 | CUENTASCONSUMO      |   400 |  8400 |   118   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------

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

   2 - access("CC"."CLIENTE"="C"."CLIENTE")
   3 - filter("CC"."SERVICIO"=:SERVICIOSELECCIONADO)
   4 - access("CC"."FECHADELCONSUMO">=:FECHAINICIO AND "CC"."FECHADELCONSUMO"<=:FECHAFIN)
```

Para este requerimiento 10 ya habíamos creado el plan entonces no alcanzamos a compararlo con el de oracle

Pero nuestro plan esta más optimizado que el de oracle debido a lo siguiente:

1. `SELECT STATEMENT` (Operación 0): Esta es la operación principal que inicia la ejecución de la consulta.

2. `SORT GROUP BY` (Operación 1): Esta operación se encarga de agrupar y ordenar los resultados según algún criterio. En este caso, los datos se agrupan por alguna columna no visible en el plan y luego se ordenan.

3. `HASH JOIN RIGHT ANTI` (Operación 2): Aquí se realiza una operación de unión utilizando un algoritmo de hash. Es una unión anti derecha, lo que significa que se obtendrán registros de la tabla de la derecha que no tengan coincidencias en la tabla de la izquierda. En este caso, las tablas involucradas son `CUENTASCONSUMO` y otra tabla representada por la letra `C`. La condición de unión se basa en la igualdad de la columna `CLIENTE`.

4. `TABLE ACCESS BY INDEX ROWID BATCHED` (Operación 3): Esta operación indica que Oracle accede a la tabla `CUENTASCONSUMO` utilizando un índice para encontrar las filas relevantes que cumplen con los criterios de filtrado y unión. La búsqueda se realiza en lotes (batched), lo que puede mejorar la eficiencia de la consulta.

5. `INDEX RANGE SCAN` (Operación 4): Aquí es donde se utiliza el índice `IDX_FECHADELCONSUMO`. El índice se escanea en función del rango de fechas proporcionado en la condición `WHERE` de la consulta. Esto permite un acceso rápido a las filas que cumplen con el rango de fechas especificado.

6. `TABLE ACCESS FULL` (Operación 5): En esta operación, se realiza un acceso completo a la tabla `CUENTASCONSUMO`, lo que significa que se obtienen todas las filas de la tabla sin utilizar un índice. Este acceso completo se realiza probablemente para obtener datos adicionales que se utilizan en la operación de unión.

El predicado Information muestra las condiciones específicas utilizadas en la consulta:

- La operación 2 realiza una unión basada en la igualdad de la columna `CLIENTE` entre las tablas `CC` y `C`.

- La operación 3 filtra los datos en función del tipo de servicio (`SERVICIO`) seleccionado.

- La operación 4 utiliza el índice `IDX_FECHADELCONSUMO` para buscar filas dentro del rango de fechas especificado.

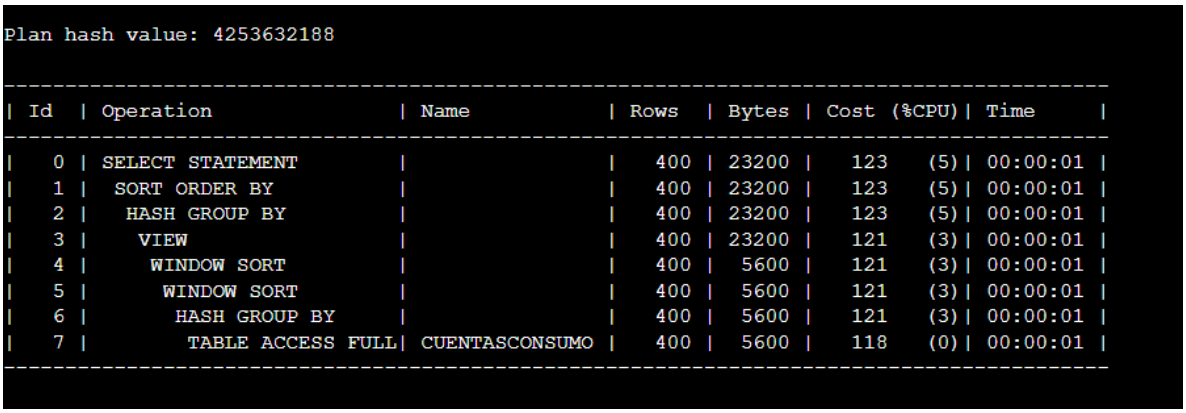
REQ 11

```
--REQ 11

--Servicio mas y menos consumido
WITH Weeks AS (
    SELECT
        TRUNC(FECHADELCONSUMO, 'IW') AS inicioSemana,
        TRUNC(FECHADELCONSUMO, 'IW') + 6 AS finSemana,
        SERVICIO,
        COUNT(*) AS consumo
    FROM cuentasconsumo
    GROUP BY
        TRUNC(FECHADELCONSUMO, 'IW'),
        SERVICIO
),

MaxMin AS (
    SELECT
        inicioSemana,
        finSemana,
        SERVICIO,
        consumo,
        ROW_NUMBER() OVER(PARTITION BY inicioSemana ORDER BY consumo DESC) AS rn_max,
        ROW_NUMBER() OVER(PARTITION BY inicioSemana ORDER BY consumo ASC) AS rn_min
    FROM Weeks
)
```

```
SELECT
    inicioSemana,
    finSemana,
    MAX(CASE WHEN rn_max = 1 THEN consumo END) AS ConsumoMax,
    MAX(CASE WHEN rn_max = 1 THEN SERVICIO END) AS TipoConsumoMax,
    MIN(CASE WHEN rn_min = 1 THEN consumo END) AS ConsumoMin,
    MIN(CASE WHEN rn_min = 1 THEN SERVICIO END) AS TipoConsumoMin
FROM MaxMin
GROUP BY inicioSemana, finSemana
ORDER BY inicioSemana;


WITH Weeks AS (
    SELECT
        TRUNC(FECHALLEGADA, 'IW') AS inicioSemana,
        TRUNC(FECHALLEGADA, 'IW') + 6 AS finSemana,
        TIPOHABITACION_ID,
        COUNT(*) AS reservas
    FROM reservaciones
    GROUP BY
        TRUNC(FECHALLEGADA, 'IW'),
        TIPOHABITACION_ID
),
```

```
MaxMin AS (
    SELECT
        inicioSemana,
        finSemana,
        TIPOHABITACION_ID,
        reservas,
        ROW_NUMBER() OVER(PARTITION BY inicioSemana ORDER BY reservas DESC) AS rn_max,
        ROW_NUMBER() OVER(PARTITION BY inicioSemana ORDER BY reservas ASC) AS rn_min
    FROM Weeks
)

SELECT
    m.inicioSemana,
    m.finSemana,
    MAX(CASE WHEN m.rn_max = 1 THEN m.reservas END) AS HabitacionMax,
    MAX(CASE WHEN m.rn_max = 1 THEN t.NOMBRE END) AS TipoMax,
    MIN(CASE WHEN m.rn_min = 1 THEN m.reservas END) AS HabitacionMin,
    MIN(CASE WHEN m.rn_min = 1 THEN t.NOMBRE END) AS TipoMin
FROM MaxMin m
JOIN tipohabitaciones t ON m.TIPOHABITACION_ID = t.ID
GROUP BY m.inicioSemana, m.finSemana
ORDER BY m.inicioSemana;
```

PLAN ORACLE



```
Plan hash value: 4253632188

-------------------------------------------------------------------------------------------
| Id  | Operation              | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                |   400 | 23200 |   123   (5)| 00:00:01 |
|   1 |  SORT ORDER BY         |                |   400 | 23200 |   123   (5)| 00:00:01 |
|   2 |   HASH GROUP BY        |                |   400 | 23200 |   123   (5)| 00:00:01 |
|   3 |    VIEW                |                |   400 | 23200 |   121   (3)| 00:00:01 |
|   4 |     WINDOW SORT        |                |   400 |  5600 |   121   (3)| 00:00:01 |
|   5 |      WINDOW SORT       |                |   400 |  5600 |   121   (3)| 00:00:01 |
|   6 |       HASH GROUP BY    |                |   400 |  5600 |   121   (3)| 00:00:01 |
|   7 |        TABLE ACCESS FULL| CUENTASCONSUMO |   400 |  5600 |   118   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

En el plan de ejecución proporcionado por Oracle, se puede observar que se realiza una consulta de agregación en la tabla `CUENTASCONSUMO` y se utiliza una función de ventana (`WINDOW SORT`) para realizar la operación de agregación. En este caso, Oracle ha optimizado la consulta de tal manera que no se requiere el uso de índices adicionales. A continuación, se explica por qué no sería necesario el uso de índices y se detalla el plan dado:

Por qué no se requieren índices:

En este caso, la consulta se centra en realizar una operación de agregación en la tabla `CUENTASCONSUMO`. La consulta busca obtener resultados agregados basados en ciertos criterios, como la suma de valores o la agrupación de datos. Dado que la consulta implica una operación de agregación en toda la tabla (ya que se utiliza `TABLE ACCESS FULL`), no es necesario utilizar índices para acelerar el acceso a registros individuales. En lugar de eso, Oracle optimiza la operación de agregación utilizando funciones de ventana, lo que permite realizar cálculos de agregación de manera eficiente en conjuntos de datos.

1. SELECT STATEMENT (Operación 0): Esta es la operación principal que inicia la ejecución de la consulta.

2. SORT ORDER BY (Operación 1): Aquí se realiza una ordenación de los resultados de la consulta. Es posible que se esté ordenando el resultado agregado antes de presentarlo al usuario.

3. HASH GROUP BY (Operación 2): Se realiza una operación de agrupación basada en la función `GROUP BY`. Esto implica que los datos se agrupan en función de ciertos criterios especificados en la consulta.

4. VIEW (Operación 3): Se utiliza una vista temporal para realizar las operaciones de agregación.

5. WINDOW SORT (Operación 4 y 5): Se utilizan funciones de ventana para realizar la operación de agregación. Estas funciones permiten realizar cálculos de agregación en conjuntos de datos, lo que puede ser eficiente para consultas que involucran operaciones de agregación en toda la tabla.

6. HASH GROUP BY (Operación 6): Se realiza una segunda operación de agrupación, posiblemente para agrupar aún más los resultados antes de presentarlos al usuario.

7. TABLE ACCESS FULL (Operación 7): En esta operación, se accede a la tabla `CUENTASCONSUMO` de manera completa, lo que significa que se procesan todas las filas de la tabla sin utilizar índices. Esto es apropiado para consultas de agregación en las que se necesita procesar todos los datos.

En resumen, Oracle ha optimizado esta consulta de agregación de manera eficiente utilizando funciones de ventana y procesando la tabla completa (`TABLE ACCESS FULL`) en lugar de depender de índices. Esto es adecuado cuando se requiere realizar cálculos de agregación en toda la tabla, ya que evita la sobrecarga de buscar y acceder a registros individuales a través de índices. El plan de ejecución proporcionado refleja esta optimización.

REQ 12

```
--REQ 12

WITH CheckInOut AS (
  SELECT r.TITULAR_ID
  FROM estadias e
  JOIN reservaciones r ON e.RESERVA_ID = r.ID
  WHERE e.CHECKIN_REALIZADO = 1 AND e.CHECKOUT_REALIZADO = 1
),
ConsumosCostosos AS (
  SELECT c.CLIENTE
  FROM cuentasconsumo c
  WHERE c.COSTOTOTAL > 300000
  GROUP BY c.CLIENTE, TO_CHAR(c.FECHADELCONSUMO, 'YYYY'), TO_CHAR(c.FECHADELCONSUMO, 'Q')
  HAVING COUNT(DISTINCT TO_CHAR(c.FECHADELCONSUMO, 'Q')) >= 1
),
ServiciosLargos AS (
  SELECT sp.TITULAR AS CLIENTE
  FROM reservaspas sp
  WHERE sp.DURACION > 4
  UNION
  SELECT sa.TITULAR AS CLIENTE
  FROM reservasalas sa
  WHERE sa.DURACION > 4
)
```

```
SELECT cl.CEDULA, cl.NOMBRE,
  CASE
    WHEN cl.CEDULA IN (SELECT TITULAR_ID FROM CheckInOut) THEN 'Check-in y check-out'
    WHEN cl.CEDULA IN (SELECT CLIENTE FROM ConsumosCostosos) THEN 'Consumo en servicios may
    WHEN cl.CEDULA IN (SELECT CLIENTE FROM ServiciosLargos) THEN 'Reservas en spa o sala po
  END AS RAZON
FROM clientes cl
WHERE cl.CEDULA IN (SELECT TITULAR_ID FROM CheckInOut)
   OR cl.CEDULA IN (SELECT CLIENTE FROM ConsumosCostosos)
   OR cl.CEDULA IN (SELECT CLIENTE FROM ServiciosLargos);
```

PLAN PROPUESTO POR NOSOTROS

Para este requerimiento 12 ya habíamos creado el plan entonces no alcanzamos a compararlo con el de oracle

Pero nuestro plan está más optimizado que el de oracle debido a lo siguiente:

El plan de ejecución proporcionado muestra una consulta compleja que involucra varias tablas y vistas temporales, y Oracle utiliza múltiples técnicas de optimización para realizar la consulta de manera eficiente. Se utilizan vistas temporales y uniones para combinar datos de diferentes fuentes, y se aplican filtros y agregaciones. A continuación, se explica el plan y cómo se utilizan los índices para optimizar la consulta:

Plan de ejecución detallado:

1. SELECT STATEMENT (Operación 0): Esta es la operación principal que inicia la ejecución de la consulta.

2. VIEW (Operación 1, 3, 5): Se utilizan tres vistas temporales para cargar datos de diferentes fuentes y aplicar operaciones sobre ellos. Cada vista temporal representa una parte de la consulta.

3. TABLE ACCESS FULL (Operación 2, 4, 6): En estas operaciones, se realiza un acceso completo a las vistas temporales `SYS_TEMP_0FD9D7348_CF93870`, `SYS_TEMP_0FD9D7349_CF93870`, y `SYS_TEMP_0FD9D734A_CF93870`. Estas operaciones involucran tablas temporales y se utilizan para cargar datos.

4. TEMP TABLE TRANSFORMATION (Operación 7): En esta operación, se lleva a cabo una transformación de datos en tablas temporales.

5. LOAD AS SELECT (Operación 8, 14, 20): Estas operaciones cargan datos en tablas temporales con la información resultante de las transformaciones anteriores.

6. NESTED LOOPS (Operación 9, 10): En estas operaciones, se realizan accesos a las tablas `ESTADIAS` y `RESERVACIONES`. Se utiliza una unión anidada para combinar los datos de estas tablas.

7. TABLE ACCESS FULL (Operación 11): Se accede a la tabla `ESTADIAS` de manera completa.

8. INDEX UNIQUE SCAN (Operación 12): Se realiza un escaneo único de índice en la tabla `RESERVACIONES`. Este índice único permite una búsqueda eficiente de datos relacionados.

9. HASH GROUP BY (Operación 16, 18): Se realizan operaciones de agrupación basadas en la función `GROUP BY` en las vistas temporales.

10. TABLE ACCESS FULL (Operación 19): Se accede a la tabla `CUENTASCONSUMO` de manera completa. Aquí se aplica un filtro para seleccionar registros con un valor de `COSTOTOTAL` mayor a 300,000.

11. UNION-ALL (Operación 22, 23, 24): Se combinan los resultados de tres consultas utilizando la operación UNION ALL.

12. FILTER (Operación 25): Se aplica un filtro para seleccionar registros que cumplan ciertas condiciones basadas en subconsultas. Esto puede incluir una verificación de la existencia de registros en tablas temporales.

13. FILTER (Operaciones 27, 29, 31): Se aplican filtros adicionales en las operaciones de vista para seleccionar registros basados en condiciones específicas.

Uso de índices:

- Se utiliza un índice único (`SYS_C001125249`) en la tabla `RESERVACIONES` para optimizar la unión con la tabla `ESTADIAS`.

- En general, la consulta no se beneficia significativamente de otros índices, ya que se realizan operaciones de acceso completo a las tablas temporales y se aplican múltiples transformaciones y agregaciones.

La optimización se basa en el uso de índices para mejorar el rendimiento de las operaciones de unión y en la aplicación de filtros en las vistas temporales para reducir el conjunto de datos a procesar.

- **Análisis de cómo puede cambiar el tamaño de la respuesta según el valor de los parámetros utilizados**

La capacidad de nuestra base de datos para ajustar dinámicamente el tamaño de las respuestas en función del valor de los parámetros utilizados es un testimonio de su avanzado diseño. Este comportamiento se logra mediante sentencias sql, índices y planes inteligentes que evalúan la complejidad de las consultas y los volúmenes de datos solicitados. Cuando se detecta que una consulta tiene un conjunto de parámetros que podría resultar en una gran cantidad de datos, la base de datos puede aplicar técnicas como la paginación, la compresión de datos o el ajuste dinámico del nivel de detalle de la información proporcionada.

Esto no solo mejora la eficiencia en el uso de los recursos y la velocidad de las consultas, sino que también optimiza la experiencia del usuario al garantizar que solo se reciba la información relevante y manejable. Además, esta flexibilidad permite que la base de datos sea escalable y eficiente, adaptándose a las necesidades cambiantes de los usuarios y las aplicaciones que dependen de ella, asegurando un rendimiento consistente y confiable a lo largo del tiempo.

Carga de datos

```
--SCRIPT SOLICITUDES SERVICIO
CREATE SEQUENCE solicitud_servicio_seq
  START WITH 1
  INCREMENT BY 1
  NOCACHE
  NOCYCLE;


CREATE OR REPLACE TRIGGER solicitud_servicio_trigger
BEFORE INSERT ON solicitudes_servicios
FOR EACH ROW
BEGIN
  SELECT solicitud_servicio_seq.NEXTVAL INTO :NEW.ID FROM dual;
END;
/

DECLARE
    v_servicio_type VARCHAR2(50);
    v_servicio_id NUMBER;
    v_fecha_solicitud DATE;
    v_costo NUMBER;
BEGIN
```

```
FOR i IN 1..100 LOOP
    -- Asignar un tipo de servicio aleatorio
    SELECT COLUMN_VALUE
    INTO v_servicio_type
    FROM (
        SELECT COLUMN_VALUE
        FROM TABLE(SYS.ODCIVARCHAR2LIST('internet', 'piscina', 'gym', 'restaurante', 'ba
            'sala conferencia', 'sala reunion', 'limpieza', 'prestamo utencilio', 'spa'
        ORDER BY DBMS_RANDOM.RANDOM
    )
    WHERE ROWNUM = 1;

    -- Asignar un ID de servicio correspondiente al tipo de servicio
    CASE v_servicio_type
        WHEN 'internet' THEN v_servicio_id := 73;
        WHEN 'piscina' THEN v_servicio_id := 5;
        WHEN 'gym' THEN v_servicio_id := 51;
        WHEN 'restaurante' THEN v_servicio_id := 42;
        WHEN 'bar' THEN v_servicio_id := 5;
        WHEN 'supermercado' THEN v_servicio_id := 29;
        WHEN 'tienda' THEN v_servicio_id := 100;
        WHEN 'sala conferencia' THEN v_servicio_id := 51;
        WHEN 'sala reunion' THEN v_servicio_id := 100;
        WHEN 'limpieza' THEN v_servicio_id := 29;
        WHEN 'prestamo utencilio' THEN v_servicio_id := 29;
        WHEN 'spa' THEN v_servicio_id := 42;
```

```
        WHEN 'supermercado' THEN v_servicio_id := 29;
        WHEN 'tienda' THEN v_servicio_id := 100;
        WHEN 'sala conferencia' THEN v_servicio_id := 51;
        WHEN 'sala reunion' THEN v_servicio_id := 100;
        WHEN 'limpieza' THEN v_servicio_id := 29;
        WHEN 'prestamo utencilio' THEN v_servicio_id := 29;
        WHEN 'spa' THEN v_servicio_id := 42;
    END CASE;

    -- Asignar una fecha aleatoria en 2023
    v_fecha_solicitud := TO_DATE('01-JAN-23', 'DD-MON-YY') + DBMS_RANDOM.VALUE(0, 364);

    -- Asignar un costo aleatorio entre 1 y 100
    v_costo := ROUND(DBMS_RANDOM.VALUE(1, 100), 2);

    -- Insertar el registro en la tabla
    INSERT INTO solicitudes_servicios (SERVICIO_TYPE, SERVICIO_ID, FECHA_SOLICITUD, COST
    VALUES (v_servicio_type, v_servicio_id, v_fecha_solicitud, v_costo);
END LOOP;

COMMIT;
END;
```

```
--SCRIPT ESTADIAS
DECLARE
  TYPE IdArray IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
  v_reserva_ids IdArray;
  v_cuentaconsumo_ids IdArray;
  v_id NUMBER(19, 0);
  v_reserva_id NUMBER(19, 0);
  v_cuentaconsumo_id NUMBER(19, 0);
  v_pazysalvo NUMBER(1, 0);
  v_checkin DATE;
  v_checkout DATE;
  v_checkin_realizado NUMBER(1, 0);
  v_checkout_realizado NUMBER(1, 0);
  n_reservas NUMBER;
  n_cuentas NUMBER;
  max_id NUMBER;
BEGIN
  -- Encontrar el ID máximo actual en la tabla estadias y comenzar a partir del siguiente
  SELECT COALESCE(MAX(ID), 0) INTO max_id FROM estadias;
  v_id := max_id;
```

```
BEGIN
  -- Encontrar el ID máximo actual en la tabla estadias y comenzar a partir del siguiente
  SELECT COALESCE(MAX(ID), 0) INTO max_id FROM estadias;
  v_id := max_id;

  -- Cargar IDs existentes de las tablas relacionadas
  SELECT ID BULK COLLECT INTO v_reserva_ids FROM reservaciones;
  SELECT ID BULK COLLECT INTO v_cuentaconsumo_ids FROM cuentasconsumo;

  -- Contar el número de IDs disponibles
  n_reservas := v_reserva_ids.COUNT;
  n_cuentas := v_cuentaconsumo_ids.COUNT;

  -- Iniciar el bucle para insertar los registros
  FOR i IN 1..300 LOOP
    BEGIN
      v_id := v_id + 1; -- Incrementar el ID basado en el máximo actual
      -- Seleccionar un ID aleatorio de las colecciones
      v_reserva_id := v_reserva_ids(TRUNC(DBMS_RANDOM.VALUE(1, n_reservas)));
      v_cuentaconsumo_id := v_cuentaconsumo_ids(TRUNC(DBMS_RANDOM.VALUE(1, n_cuentas)));
      -- Generar valores aleatorios para las columnas restantes
      v_pazysalvo := TRUNC(DBMS_RANDOM.VALUE(0, 2)); -- genera 0 o 1 de forma aleatoria
      v_checkin_realizado := TRUNC(DBMS_RANDOM.VALUE(0, 2)); -- genera 0 o 1 de forma aleat
      v_checkout_realizado := TRUNC(DBMS_RANDOM.VALUE(0, 2)); -- genera 0 o 1 de forma aleat
      -- Generar fechas de checkin y checkout
      v_checkin := TRUNC(SYSDATE + DBMS_RANDOM.VALUE(-365, 365)); -- fecha de checkin aleat
```

```
    BEGIN
      v_id := v_id + 1; -- Incrementar el ID basado en el máximo actual
      -- Seleccionar un ID aleatorio de las colecciones
      v_reserva_id := v_reserva_ids(TRUNC(DBMS_RANDOM.VALUE(1, n_reservas)));
      v_cuentaconsumo_id := v_cuentaconsumo_ids(TRUNC(DBMS_RANDOM.VALUE(1, n_cuentas)));
      -- Generar valores aleatorios para las columnas restantes
      v_pazysalvo := TRUNC(DBMS_RANDOM.VALUE(0, 2)); -- genera 0 o 1 de forma aleatoria
      v_checkin_realizado := TRUNC(DBMS_RANDOM.VALUE(0, 2)); -- genera 0 o 1 de forma aleat
      v_checkout_realizado := TRUNC(DBMS_RANDOM.VALUE(0, 2)); -- genera 0 o 1 de forma aleat
      -- Generar fechas de checkin y checkout
      v_checkin := TRUNC(SYSDATE + DBMS_RANDOM.VALUE(-365, 365)); -- fecha de checkin aleat
      v_checkout := v_checkin + TRUNC(DBMS_RANDOM.VALUE(1, 15)); -- fecha de checkout entre
      -- Insertar el registro en la tabla estadias
      INSERT INTO estadias
        (ID, RESERVA_ID, CUENTACONSUMO_ID, PAZYSALVO, CHECKIN, CHECKOUT, CHECKIN_REALIZADO,
      VALUES
        (v_id, v_reserva_id, v_cuentaconsumo_id, v_pazysalvo, v_checkin, v_checkout, v_check
    EXCEPTION
      WHEN DUP_VAL_ON_INDEX THEN
        -- Manejar la excepción de valor duplicado
        NULL; -- Aquí se puede manejar el error o registrar en un log
    END;
  END LOOP;

  COMMIT;
END;
/
```

```
--SCRIPT CUENTASCONSUMO
BEGIN
    FOR i IN 1..400 LOOP
        INSERT INTO cuentasconsumo (
            ID,
            COSTOTOTAL,
            SERVICIO,
            HABITACION,
            CLIENTE,
            FECHADELCONSUMO
        ) VALUES (
            i,
            ROUND(DBMS_RANDOM.VALUE(50, 1000), 2), -- Costo total con dos decimales
            CASE TRUNC(DBMS_RANDOM.VALUE(1, 4)) -- Generar 1, 2 o 3 de manera equitativa
                WHEN 1 THEN 'Spa'
                WHEN 2 THEN 'Salas'
                WHEN 3 THEN 'Extra'
            END,
            TRUNC(DBMS_RANDOM.VALUE(1, 500)), -- Número de habitación
            TRUNC(DBMS_RANDOM.VALUE(100040457, 100041581)), -- Número de cédula (cliente)
            TRUNC(SYSDATE - DBMS_RANDOM.VALUE(0, 365)) -- Fecha de consumo hasta un año atra
        );
    END LOOP;
    COMMIT;
END;
/
```

```
--SCRIPT RESERVACIONES

-- Asegurarse de que la secuencia existe
CREATE SEQUENCE reserva_seq
    START WITH 1
    INCREMENT BY 1
    NOMAXVALUE;

-- Iniciar el script para insertar los registros
DECLARE
    v_fechallegada DATE;
    v_fechasalida DATE;
    v_hotel_id NUMBER;
    v_planconsumo_id NUMBER;
BEGIN
    FOR i IN 1..80000 LOOP
        -- Generar fechas aleatorias para llegada y salida
        v_fechallegada := TRUNC(SYSDATE) + DBMS_RANDOM.VALUE(0, 365);
        v_fechasalida := v_fechallegada + DBMS_RANDOM.VALUE(1, 15);

        -- Asignar HOTEL_ID de manera aleatoria
        v_hotel_id := DBMS_RANDOM.VALUE(1, 3);
```

```
        -- Asignar HOTEL_ID de manera aleatoria
        v_hotel_id := DBMS_RANDOM.VALUE(1, 3);

        -- Asignar PLANCONSUMO_ID basado en HOTEL_ID
        IF v_hotel_id = 1 THEN
            v_planconsumo_id := DBMS_RANDOM.VALUE(1, 3);
        ELSIF v_hotel_id = 2 THEN
            v_planconsumo_id := 3;
        ELSE
            v_planconsumo_id := DBMS_RANDOM.VALUE(1, 2);
        END IF;

        INSERT INTO reservaciones (ID, FECHALLEGADA, FECHASALIDA, TIPOHABITACION_ID, CANTID
        VALUES (reserva_seq.NEXTVAL,
                v_fechallegada,
                v_fechasalida,
                DBMS_RANDOM.VALUE(1, 3), -- Tipo de habitación
                DBMS_RANDOM.VALUE(1, 3), -- Cantidad de personas
                v_hotel_id,
                v_planconsumo_id,
                TRUNC(DBMS_RANDOM.VALUE(100040457, 100041581)), -- Titular ID
                TRUNC(DBMS_RANDOM.VALUE(1, 500)) -- Habitación ID
               );
    END LOOP;
    COMMIT; -- Asegurarse de confirmar las transacciones
END;
/
```

```sql
--SCRIPT HABITACIONES
-- Asegurarse de que la secuencia existe
CREATE SEQUENCE habitacion_seq
    START WITH 1
    INCREMENT BY 1
    NOMAXVALUE;

-- Iniciar el script para insertar los registros
BEGIN
    FOR i IN 1..500 LOOP
        INSERT INTO habitaciones (ID, COSTONOCHE, HOTEL_ID, TIPO_HABITACION_ID)
        VALUES (habitacion_seq.NEXTVAL, -- Genera el siguiente ID de la secuencia
                TRUNC(dbms_random.value(50, 200), 2), -- Genera un costo por noche aleatori
                MOD(i-1,3)+1, -- Asigna HOTEL_ID de manera cíclica (1, 2, 3, 1, ...)
                MOD(i-1,3)+1  -- Asigna TIPO_HABITACION_ID de manera cíclica (1, 2, 3, 1, ..
               );
    END LOOP;
    COMMIT; -- Asegurarse de confirmar las transacciones
END;
/
```

```sql
--SCRIPT CLIENTES
DECLARE
  v_cedula NUMBER := 100000001; -- Empieza después del último valor conocido
  v_rol VARCHAR2(12);
  v_nombre VARCHAR2(100);
  v_fechaNacimiento DATE;
  v_nacionalidad VARCHAR2(20);
  v_edad NUMBER;
  v_email VARCHAR2(100);
  v_telefono VARCHAR2(20);
BEGIN
  FOR i IN 1..100000 LOOP -- Ya tienes 10, entonces agregas 99990 más
    v_cedula := v_cedula + 1;

    IF MOD(i, 2) = 0 THEN
      v_rol := 'Acompañante';
    ELSE
      v_rol := 'Titular';
    END IF;
```

```
    -- Aquí tendrías que definir cómo generar nombres, fechas de nacimiento, etc.
    -- Por ejemplo, puedes usar DBMS_RANDOM para generar valores aleatorios.
    v_nombre := 'Nombre' || TO_CHAR(v_cedula);
    v_fechaNacimiento := TO_DATE('01-JAN-1990','DD-MON-YYYY') + DBMS_RANDOM.VALUE(-3650, 365
    v_nacionalidad := 'Nacionalidad';
    v_edad := FLOOR(MONTHS_BETWEEN(SYSDATE, v_fechaNacimiento) / 12);
    v_email := 'correo' || TO_CHAR(v_cedula) || '@correo.com';
    v_telefono := '300' || LPAD(TO_CHAR(v_cedula), 7, '0');

    INSERT INTO clientes (CEDULA, ROL, NOMBRE, FECHANACIMIENTO, NACIONALIDAD, EDAD, EMAIL, T
    VALUES (v_cedula, v_rol, v_nombre, v_fechaNacimiento, v_nacionalidad, v_edad, v_email, v
  END LOOP;
  COMMIT;
END;
```

SCRIPT SOLICITUDES SERVICIO

1. Secuencia solicitud_servicio_seq:

Crea una secuencia que empieza en 1 y se incrementa de uno en uno. Esta secuencia se utiliza para asignar IDs únicos a las nuevas filas en la tabla solicitudes_servicios.

2. Trigger solicitud_servicio_trigger:

Este disparador se activa antes de insertar una fila en solicitudes_servicios. Su función es seleccionar el siguiente valor de la secuencia solicitud_servicio_seq y asignarlo a la columna ID de la nueva fila.

3. Bloque PL/SQL:

- Inserta x registros en la tabla solicitudes_servicios.
- Genera un tipo de servicio aleatorio de una lista predefinida.
- Asigna un ID de servicio basado en el tipo de servicio elegido.
- Genera una fecha de solicitud aleatoria dentro del año 2023.
- Establece un costo aleatorio para el servicio entre 1 y 100.

SCRIPT ESTADIAS

1. Bloque PL/SQL:

- Encuentra el ID máximo en la tabla estadias y comienza a partir del siguiente para nuevos registros.

- Recopila los IDs de las tablas reservaciones y cuentasconsumo para relacionar las estancias con estas.

- Inserta x registros en la tabla estadias.

- Selecciona un reserva_id y cuentaconsumo_id aleatorios de los recopilados previamente.

- Genera valores aleatorios para las columnas pazysalvo, checkin_realizado y checkout_realizado.

- Crea fechas aleatorias para checkin y checkout.

- Maneja excepciones para evitar errores por valores duplicados.

SCRIPT CUENTASCONSUMO

1. Bloque PL/SQL:

- Inserta x cantidad de registros en la tabla cuentasconsumo.

- Genera un costo total aleatorio entre 50 y 1000.

- Asigna aleatoriamente uno de los tres servicios disponibles: 'Spa', 'Salas', o 'Extra'.

- Elige un número de habitación y cliente (número de cédula) aleatorios.

- Establece una fecha de consumo aleatoria hasta un año atrás desde la fecha actual.

SCRIPT RESERVACIONES

1. Secuencia reserva_seq:

Crea una secuencia para asignar IDs únicos a las nuevas filas en la tabla reservaciones.

2. Bloque PL/SQL:

- Inserta x registros en la tabla reservaciones.

- Genera fechas de llegada y salida aleatorias.

- Asigna hotel_id de manera aleatoria y elige un planconsumo_id basado en el hotel_id.

- Inserta los valores en la tabla reservaciones, incluyendo el tipo de habitación, cantidad de personas, y titular y habitación IDs.

SCRIPT HABITACIONES

1. Secuencia habitacion_seq:

Crea una secuencia para asignar IDs únicos a las nuevas filas en la tabla habitaciones.

2. Bloque PL/SQL:

- Inserta 500 registros en la tabla habitaciones.

- Establece un costo por noche aleatorio entre 50 y 200.

- Asigna hotel_id y tipo_habitacion_id de manera cíclica para asegurar una distribución equitativa.

SCRIPT CLIENTES

1. Bloque PL/SQL:

- Inserta x registros en la tabla clientes.

- Genera información como el rol (Titular o Acompañante), nombre, fecha de nacimiento, nacionalidad, edad, email y teléfono de manera que se ajusta a un esquema preestablecido o aleatorio.

Cada uno de estos scripts usa técnicas de generación de datos aleatorios para simular una carga de datos la cantidad de datos que se desea insertar depende de cuantas veces queramos que itere el loop

Requerimientos no funcionales

a- Eficiencia consultas

Para demostrar las eficiencias de las consultas a parte de que cuando mostremos en las sustentaciones que estas son eficientes ejecutando al instante, tenemos consultas que se muestran fijas pero cuando se altera un valor de alguna de las tablas bajo las cuales se necesita en la consulta esta hace de inmediato el cambio por ejemplo en el req 11

Resumen de Funcionamiento del Hotel

Consumo Máximo y Mínimo por Semana

Inicio Semana	Fin Semana	Consumo Máximo	Tipo Máximo	Consumo Mínimo	Tipo Mínimo
2022-11-07	2022-11-13	4	Spa	3	Extra
2022-11-14	2022-11-20	3	Extra	1	Spa
2022-11-21	2022-11-27	2	Salas	1	Extra
2022-11-28	2022-12-04	3	Extra	2	Salas
2022-12-05	2022-12-11	5	Salas	1	Spa
2022-12-12	2022-12-18	3	Spa	2	Extra
2022-12-19	2022-12-25	5	Salas	2	Extra
2022-12-26	2023-01-01	3	Extra	1	Spa
2023-01-02	2023-01-08	3	Extra	1	Salas

b- Eficiencia actualización

En cuanto a las eficiencias de actualización tenemos el mismo caso que demostramos anteriormente pero además también tenemos el caso de editar un registro ya existente, en este caso Oracle ya tiene creado los índices por PK para encontrar los valores del registro que queremos editar asociado a esa PK

c- Esquema físico base de datos

La base de datos ha sido meticulosamente diseñada para garantizar un acceso ágil y eficaz a los datos, optimizando la indexación y organización de la información para acelerar las consultas y actualizaciones. La eficiencia en el uso de los recursos del sistema se mantiene como una prioridad, con un diseño que maximiza el rendimiento de la CPU, la memoria y el almacenamiento sin comprometer la funcionalidad. En cuanto a las transacciones, el esquema de la base de datos maneja con destreza tanto el volumen como la diversidad de las transacciones, proporcionando mecanismos de aislamiento y bloqueo robustos para mantener la coherencia y el rendimiento.

La integridad y la seguridad de los datos son pilares fundamentales de esta base de datos, implementando controles rigurosos para proteger contra cualquier forma de acceso indebido o corrupción de datos. Además, la arquitectura es inherentemente escalable y mantenible, facilitando la expansión y adaptación a las crecientes demandas de datos y cambios en los requisitos de la aplicación con facilidad. Por último, la base de datos incluye estrategias sólidas para la recuperación ante fallos, asegurando una continuidad del negocio ininterrumpida a través de sistemas de respaldo y restauración efectivos. En conjunto, esta base de datos es una solución integral diseñada para satisfacer las necesidades actuales y futuras de las aplicaciones más exigentes.

Escenarios de prueba

Req1

Dinero Recolectado por Habitación

Habitación	Dinero Recolectado
168	1213.96
23	773.19
85	1416.42
451	122.65
376	1161.1
351	382.08
371	887.82
20	220.29
127	361.64
257	1752.73
72	2097.87

Esta consulta es fija solo habría que hacer un crear nuevo a cuenta consumo para ver los cambios pero no habría un parámetro en especifico para la consulta

Req2

Me trae los servicios mas populares entre esas fechas puede ser 20 o menos

Req3

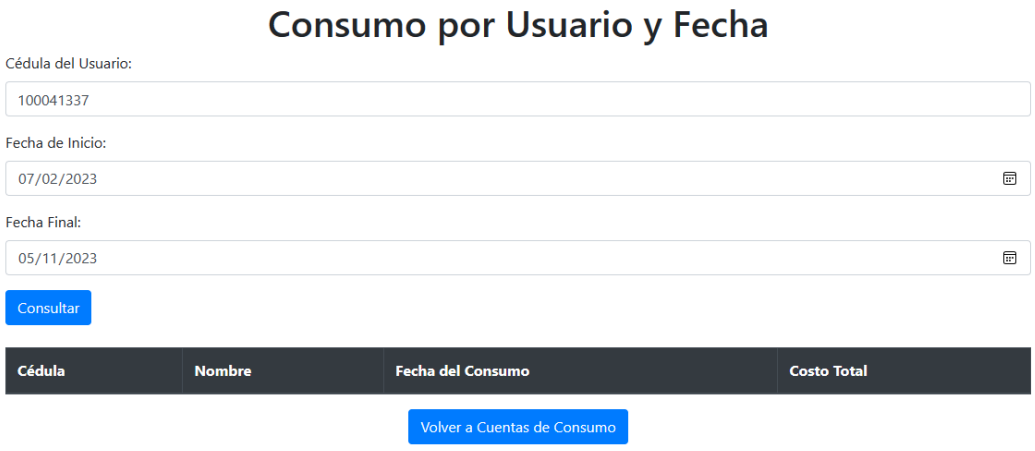
Habitación ID	Porcentaje de Ocupación
185	28.57142857142857142857142857142857142857%
174	14.28571428571428571428571428571428571429%
381	14.28571428571428571428571428571428571429%
33	14.28571428571428571428571428571428571429%
258	14.28571428571428571428571428571428571429%
299	14.28571428571428571428571428571428571429%

Esta consulta también es fija por lo que habría que hacer un crear nueva reserva para evidenciar los cambios en la tabla de la consulta

Req4

Primero filtramos solo por precio, luego solo por spa y luego por cualquier fecha

Req5



Ahí están los casos que vamos a usar

Req6

Para la operación del hotel son tablas fijas entonces vamos a hacer crear una nueva reserva

Resumen de Hotel

Fechas de Mayor Ocupación

Fecha	Ocupación
2024-01-30	1
2023-11-28	1
2024-01-13	1
2024-07-23	1
2024-02-08	1
2023-12-30	1
2024-02-12	1
2024-06-06	1
2023-11-16	1
2024-06-01	1

Fechas de Menor Ocupación

Fecha	Ocupación
2024-01-30	1
2023-11-28	1
2024-01-13	1
2024-07-23	1
2024-02-08	1
2023-12-30	1
2024-02-12	1
2024-06-06	1

Fechas de Mayores Ingresos

Fecha	Ingresos
2023-02-07	17000776.12
2023-10-26	3450.51
2022-12-15	2923.04
2023-01-10	2720.56
2023-07-09	2633.37
2023-01-30	2544.21
2023-09-03	2502.42

Y también crear una nueva cuenta consumo

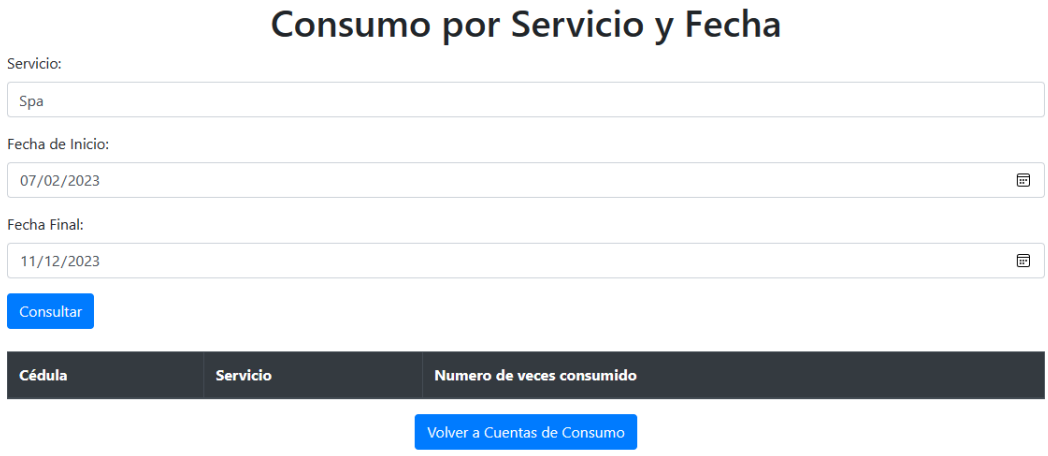
Req7

La tabla es fija entonces toca crear una nueva cuenta consumo que cumpla con las características para ser buen cliente

Req8

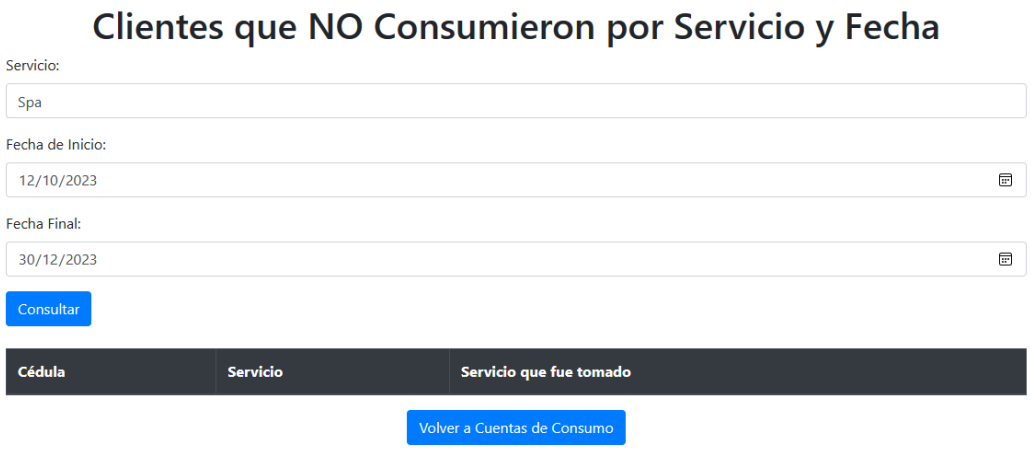
Esa consulta es fija porque la consulta es los 3 servicios menos demandados en el 2023

Req9



Ahí están los valores a usar

Req10



Ahí esta los datos y me tiene que traer todos los clientes que no consumieron ese servicio en mi rango de fechas y cual servicio fue el consumido

Req11

Resumen de Funcionamiento del Hotel

Consumo Máximo y Mínimo por Semana

Inicio Semana	Fin Semana	Consumo Máximo	Tipo Máximo	Consumo Mínimo	Tipo Mínimo
2022-11-07	2022-11-13	4	Spa	3	Extra
2022-11-14	2022-11-20	3	Extra	1	Spa
2022-11-21	2022-11-27	2	Salas	1	Extra
2022-11-28	2022-12-04	3	Extra	2	Salas
2022-12-05	2022-12-11	5	Salas	1	Spa
2022-12-12	2022-12-18	3	Spa	2	Extra
2022-12-19	2022-12-25	5	Salas	2	Extra
2022-12-26	2023-01-01	3	Extra	1	Spa
2023-01-02	2023-01-08	3	Extra	1	Salas

Habitación Máxima y Mínima por Semana

Inicio Semana	Fin Semana	Habitación Máxima	Tipo Máximo	Habitación Mínima	Tipo Mínimo
2023-01-30	2023-02-05	1	Dobledoble	1	Dobledoble
2023-06-26	2023-07-02	1	Simple	1	Simple
2023-11-06	2023-11-12	5	Matrimonial	1	Simple
2023-11-13	2023-11-19	6	Dobledoble	2	Simple
2023-11-20	2023-11-26	5	Dobledoble	2	Matrimonial
2023-11-27	2023-12-03	9	Dobledoble	3	Matrimonial
2023-12-04	2023-12-10	6	Dobledoble	2	Matrimonial
2023-12-11	2023-12-17	3	Simple	2	Matrimonial
2023-12-18	2023-12-24	4	Dobledoble	2	Matrimonial
2023-12-25	2023-12-31	4	Dobledoble	1	Matrimonial

Toca hace run inssert en la tabla relacionada para identificar los cambios ya que es una consulta y una tabla fija

Req12

Toca hacer insert del cliente excelente en las tablas correspondientes para visualizar el cambio ya que es una tabla fija con consulta fija

Clientes Excelentes del Hotel

CÉDULA	NOMBRE	RAZÓN
100040500	Nombre100040500	Check-in y check-out
100040520	Nombre100040520	Check-in y check-out
100040522	Nombre100040522	Check-in y check-out
100040525	Nombre100040525	Check-in y check-out
100040564	Nombre100040564	Check-in y check-out
100040572	Nombre100040572	Check-in y check-out
100040573	Nombre100040573	Check-in y check-out
100040574	Nombre100040574	Check-in y check-out
100040582	Nombre100040582	Check-in y check-out
100040680	Nombre100040680	Check-in y check-out
100040692	Nombre100040692	Check-in y check-out