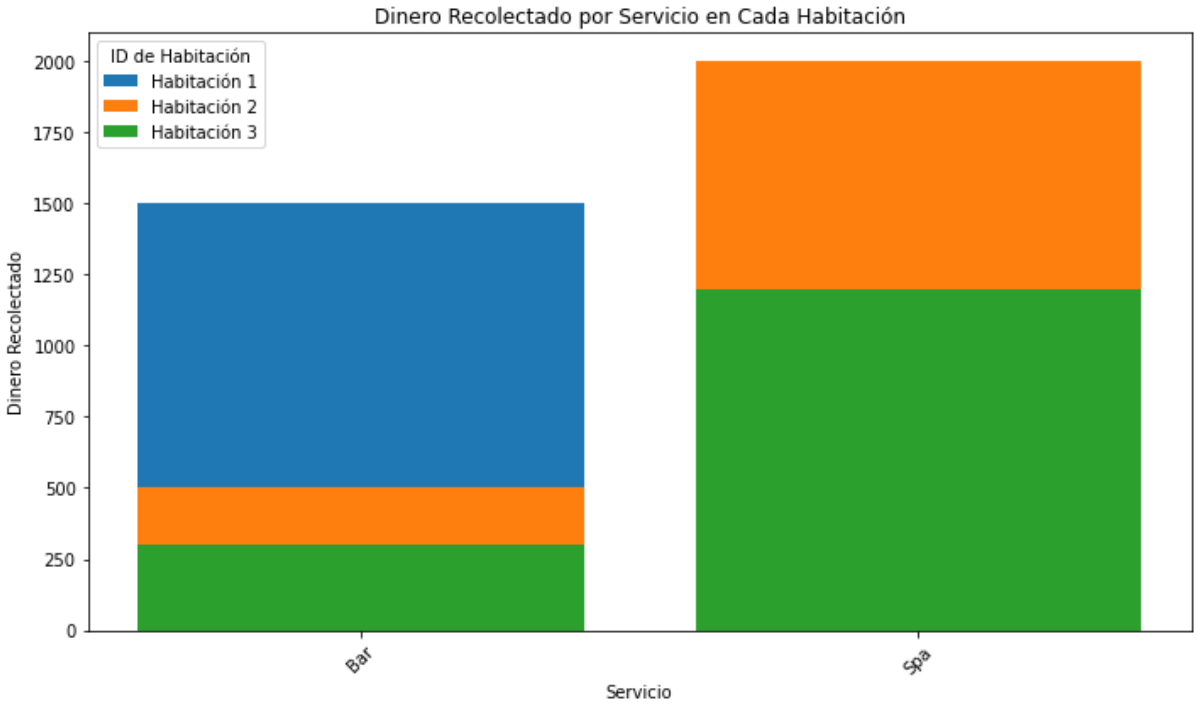
Documentación requerimientos funcionales

Requerimiento 1:
Para este requerimiento se obtiene el identificador de cada habitación (habitacion_id), el nombre de cada servicio (servicio) y calcula la suma total del precio de los servicios (dinero_recolectado). Luego decidimos unir las tablas 'habitaciones' y 'reservas' por medio de los campos 'id' y 'habitaciones_id'. También se unieron las tablas 'reservas_servicios' con 'reservas', a través de sus id's. De igual manera sucedió con las tablas 'reservas_servicios' y 'servicios'. Para después agrupar los valores por el identificador de la habitación y el nombre del servicio para sumar correctamente los precios de servicios por habitación, y luego ordenarlos por el nombre del servicio.

```
@Query(value = "SELECT\r\n" +
        "    h.id AS habitacion_id,\r\n" +
        "    s.nombre AS servicio,\r\n" +
        "    SUM(s.precio) AS dinero_recolectado\r\n" +
        "FROM\r\n" +
        "    habitaciones h\r\n" +
        "JOIN\r\n" +
        "    reservas r ON h.id = r.habitaciones_id\r\n" +
        "JOIN\r\n" +
        "    reservas_servicios rs ON r.id = rs.reservas_id\r\n" +
        "JOIN\r\n" +
        "    servicios s ON rs.servicios_id = s.id\r\n" +
        "WHERE\r\n" +
        "    r.fecha_entrada >= (SELECT CURRENT_DATE-365 from dual)\r\n" +
        "GROUP BY\r\n" +
        "    h.id, s.nombre\r\n" +
        "ORDER BY\r\n" +
        "    h.id, s.nombre"
            , nativeQuery = true )
            Collection<Object[]> darConsumos();


}
```



En esta gráfica se puede observar la distribución de los datos con respecto a los parámetros de entrada utilizados en el requerimiento funcional. En este ejemplo se muestran dos servicios, 'Bar' y 'Spa'. Cada barra representa la cantidad total de dinero recolectada por cada servicio en las tres habitaciones diferentes. Se puede observar que el servicio 'Spa' ha generado más ingresos en total en comparación con el servicio 'Bar'. Para cada servicio, las barras están segmentadas por color para representar cada habitación. Esto permite comparar directamente cuánto dinero generó cada habitación por cada servicio.

Las diferencias en la altura de los segmentos de las barras para un servicio específico pueden indicar variabilidad en la popularidad o en el precio del servicio entre las habitaciones. Si una habitación tiene una barra mucho más alta que las otras para un servicio dado, esto podría sugerir que ese servicio es particularmente popular o está a un precio más alto en esa habitación.

La selección de un rango de fechas limitado al "último año corrido" puede influir significativamente en el tamaño de la respuesta. La configuración de los datos de prueba también afectará estos resultados; un conjunto de datos con más variabilidad en las fechas de entrada o en los tipos de habitaciones podría resultar en una distribución más uniforme o más variada de los ingresos.

La selectividad de la consulta (es decir, qué tan selectiva es la consulta al filtrar los registros) cambiará el tamaño de la respuesta. Si los parámetros de entrada se ajustan para incluir más fechas o un rango de precios más amplio, el tamaño de la respuesta (el número de registros devueltos) probablemente aumentará. Si la consulta está configurada para seleccionar una gama más estrecha de fechas o precios, el resultado será más selectivo y, por ende, posiblemente más pequeño.

Plan de ejecución por Oracle/ Tiempos de ejecución:

```
Plan hash value: 285612207

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                  | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                       |     1 |   214 |     4  (25)| 00:00:01 |
|   1 |  SORT GROUP BY                |                       |     1 |   214 |     4  (25)| 00:00:01 |
|   2 |   NESTED LOOPS                |                       |     1 |   214 |     1   (0)| 00:00:01 |
|   3 |    NESTED LOOPS               |                       |     1 |   214 |     1   (0)| 00:00:01 |
|   4 |     NESTED LOOPS              |                       |     1 |   179 |     1   (0)| 00:00:01 |
|   5 |      INDEX FULL SCAN          | RESERVAS_SERVICIOS_PK |     1 |    26 |     1   (0)| 00:00:01 |
|   6 |      TABLE ACCESS BY INDEX ROWID| SERVICIOS           |     1 |   153 |     0   (0)| 00:00:01 |
|*  7 |       INDEX UNIQUE SCAN       | SERVICIOS_PK          |     1 |       |     0   (0)| 00:00:01 |
|*  8 |     INDEX UNIQUE SCAN         | RESERVAS_PK           |     1 |       |     0   (0)| 00:00:01 |
|*  9 |    TABLE ACCESS BY INDEX ROWID| RESERVAS              |     1 |    35 |     0   (0)| 00:00:01 |
|  10 |     FAST DUAL                 |                       |     1 |       |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------

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

   7 - access("RS"."SERVICIOS_ID"="S"."ID")
   8 - access("R"."ID"="RS"."RESERVAS_ID")
```

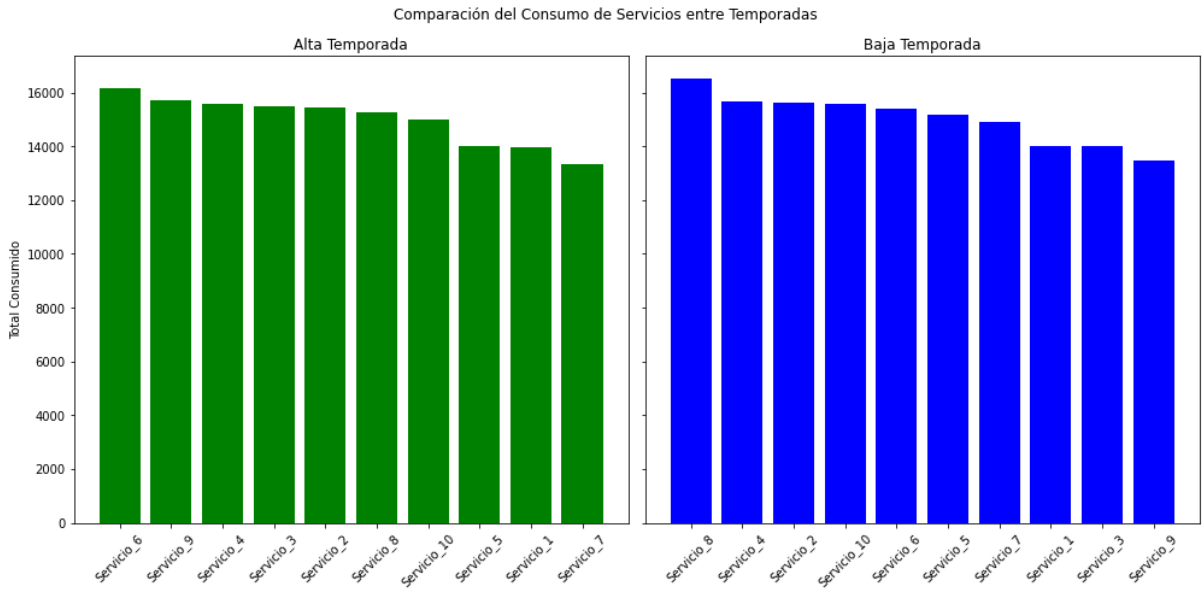
Como manera opcional se podría plantear un plan de ejecución basado en la utilización de índices. Para reducir el costo de la consulta, se podría crear un índice para 'fecha_entrada. Optimizaría las operaciones que involucren esta columna (como el filtrado en la cláusula WHERE) podrían realizarse más rápidamente. Un índice permitiría al motor de la base de datos encontrar las filas relevantes sin tener que realizar un escaneo completo de la tabla.

Requerimiento 2:

Para este requerimiento se seleccionan el nombre del servicio y la suma del consumo de los servicios en la tabla 'reserva_servicios'; Luego, se unen las tablas 'reserva_servicios' con 'servicios' y de la misma manera se unen 'reserva_servicios' con 'reservas'. Después de esto, se aplica un filtrado por rango de fechas utilizando un where, para especificar fecha inicial y fecha final; Para luego agrupar los datos obtenidos por nombre, y ordenarlos en orden descendente por valor consumido, y mostrar únicamente los primeros 20 registros.

```
@Query(value = "SELECT s.nombre AS servicio, SUM(rs.cont_servicios) AS total_consumido " +
        "FROM reservas_servicios rs " +
        "INNER JOIN servicios s ON rs.servicios_id = s.id " +
        "INNER JOIN reservas r ON rs.reservas_id = r.id " +
        "WHERE r.fecha_entrada >  TO_DATE(:fechaI, 'YYYY-MM-DD') AND r.fecha_salida <
TO_DATE(:fechaO,'YYYY-MM-DD')" +
        "GROUP BY s.nombre " +
        "ORDER BY total_consumido DESC " +
        "FETCH FIRST 20 ROWS ONLY", nativeQuery = true )
```

```
        Collection<Object[]> darRta(@Param("fechaI") String fechaInicial, @Param("fechaO") String fechaFinal);
}
```



El tamaño de la respuesta (el volumen de consumo total por servicio) cambia significativamente entre la alta y baja temporada, lo cual es esperable teniendo en cuenta la variabilidad entre temporadas. Los parámetros de entrada, en este caso, son los rangos de fechas que definen cada temporada. La configuración de estos rangos afectará directamente el tamaño de la respuesta: un rango más amplio podría aumentar el volumen de consumo total registrado, mientras que un rango más estrecho podría disminuirlo.

Plan de ejecución por Oracle/ Tiempos de ejecución:

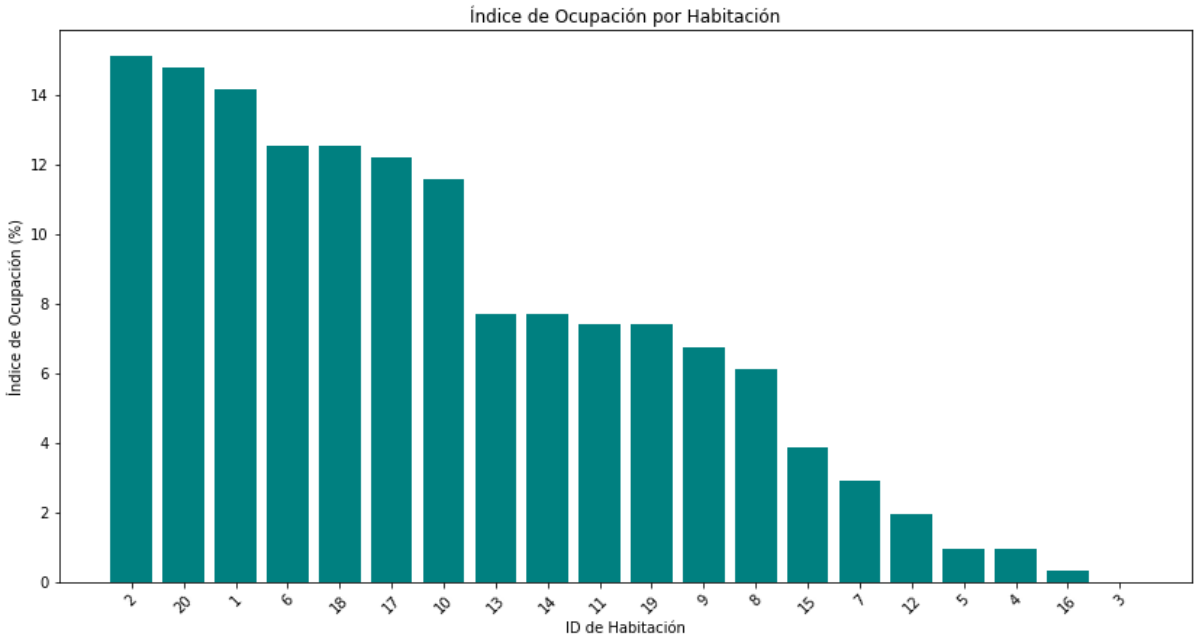
```
Plan hash value: 4160162554

-------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                     |    20 |  3320 |     5  (60)| 00:00:01 |
|   1 |  SORT ORDER BY                     |                     |    20 |  3320 |     5  (60)| 00:00:01 |
|*  2 |   VIEW                             |                     |    20 |  3320 |     4  (50)| 00:00:01 |
|*  3 |    WINDOW SORT PUSHED RANK         |                     |     1 |   210 |     4  (50)| 00:00:01 |
|   4 |     HASH GROUP BY                  |                     |     1 |   210 |     4  (50)| 00:00:01 |
|   5 |      NESTED LOOPS                  |                     |     1 |   210 |     2   (0)| 00:00:01 |
|   6 |       NESTED LOOPS                 |                     |     1 |   210 |     2   (0)| 00:00:01 |
|   7 |        NESTED LOOPS                |                     |     1 |   179 |     2   (0)| 00:00:01 |
|   8 |         TABLE ACCESS FULL          | RESERVAS_SERVICIOS  |     1 |    39 |     2   (0)| 00:00:01 |
|   9 |         TABLE ACCESS BY INDEX ROWID| SERVICIOS           |     1 |   140 |     0   (0)| 00:00:01 |
|* 10 |          INDEX UNIQUE SCAN         | SERVICIOS_PK        |     1 |       |     0   (0)| 00:00:01 |
|* 11 |        INDEX UNIQUE SCAN           | RESERVAS_PK         |     1 |       |     0   (0)| 00:00:01 |
|* 12 |       TABLE ACCESS BY INDEX ROWID  | RESERVAS            |     1 |    31 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------

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

Como manera opcional se podría plantear un plan de ejecución basado en la utilización de índices. Para reducir el costo de la consulta, se podría crear un índice para ‘fecha_entrada. Optimizaría las operaciones que involucren esta columna (como el filtrado en la cláusula WHERE) podrían realizarse más rápidamente. Un índice permitiría al motor de la base de datos encontrar las filas relevantes sin tener que realizar un escaneo completo de la tabla.

Requerimiento 3:

Para este requerimiento se selecciona el id de la habitación y realizo una operación para sacar el índice de ocupación. Esto lo modelamos a través del atributo cont_habiaciones, el cual se encarga de contabilizar la cantidad de veces que ha sido reservada una habitación, y este valor lo multiplico por 100, y lo divido entre la variable ‘Dia’, la cual corresponde al número del día dentro del año. Este valor lo sacamos por medio del controller, utilizando la función LocalDate.now() y getDayOfYear(). Así logramos obtener el índice de ocupación de cada una de las habitaciones del hotel

```
@Query(value = "select id, round(cont_habitaciones*100/ :Dia, 2) as resultado_multiplicacion\r\n" +
        "from habitaciones", nativeQuery = true )
        Collection<Object[]> darRta(@Param("Dia")Integer diaActual);
}
```



Si hay una alta variabilidad en cont_habitaciones (es decir, algunas habitaciones se reservan con mucha más frecuencia que otras), veremos una distribución amplia en el índice de ocupación. Al cambiar el valor del parámetro 'Día', el tamaño de la respuesta (es decir, el valor calculado del índice de ocupación) cambiará. A principios de año, es posible que el índice de ocupación parezca exageradamente alto, mientras que a medida que avanza el año, el índice se reducirá y podría dar una imagen más realista de la ocupación.

Plan de ejecución por Oracle/ Tiempos de ejecución:

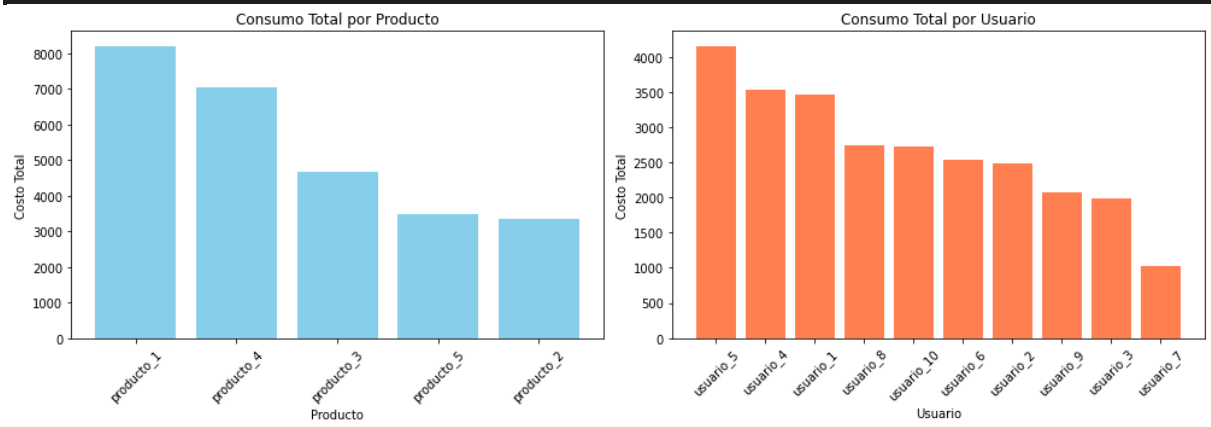
```
Plan hash value: 2883760253

-------------------------------------------------------------------------------
| Id  | Operation         | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |              |     1 |    26 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| HABITACIONES |     1 |    26 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

Para este plan de ejecución, el costo es muy bajo. Por lo que un plan de ejecución alternativo puede llegar a tener el mismo costo o más, llegando a ser no atractivo en términos de costo.

Requerimiento 5:

Para este requerimiento se unen las tablas ‘usuarios’ y ‘reservas’, a través de los campos ‘id’ y ‘usuarios_id’, de igual manera se unen las tablas ‘reservas’ y ‘reservas_servicios’, ‘servicios’ y ‘reservas_servicios’, ‘servicios’ y ‘servicios_productos’, y ‘productos’ y ‘servicios_productos’. Luego filtro los resultados por medio de las coincidencias con el login de usuario, y la fecha de inicio y finalización.

```
@Query(value = "SELECT\r\n" +
        "    u.login AS usuario,\r\n" +
        "    p.nombre AS producto_nombre,\r\n" +
        "    rfs.fecha_consumo AS fecha_consumo,\r\n" +
        "    p.costo AS costo_producto\r\n" +
        "FROM\r\n" +
        "    usuarios u\r\n" +
        "JOIN\r\n" +
        "    reservas r ON u.id = r.Usuarios_id\r\n" +
        "JOIN\r\n" +
        "    reservas_servicios rfs ON r.id = rfs.reservas_id\r\n" +
        "JOIN\r\n" +
        "    servicios s ON rfs.servicios_id = s.id\r\n" +
        "JOIN\r\n" +
        "    servicios_productos sp ON s.id = sp.servicios_id\r\n" +
        "JOIN\r\n" +
        "    productos p ON sp.productos_id = p.id\r\n" +
        "WHERE\r\n" +
        "    u.login = :login -- Reemplaza con el nombre de usuario que estás buscando\r\n" +
        "    AND r.fecha_entrada >= TO_DATE( :fechaI, 'YYYY-MM-DD')\r\n" +
        "    AND r.fecha_salida <= TO_DATE( :fechaO, 'YYYY-MM-DD')", nativeQuery = true )

        Collection<Object[]> darRta(@Param("fechaI") String fechaInicial, @Param("fechaO") String fechaFinal,
@Param(value = "login") String login);
}
```



Si algunos productos tienen barras significativamente más altas que otros, esto indica que esos productos son más populares o tienen un costo más alto. Si ciertos usuarios tienen un consumo total más alto que otros, podrían ser considerados clientes premium y podrían ser objetivos para promociones o programas de fidelización.

Si los parámetros de fecha fechaInicial y fechaFinal se ajustan para incluir diferentes rangos, como una temporada alta o baja, esto cambiará el volumen total de consumo registrado y, por ende, la altura de las barras en los gráficos. Cambiar el parámetro login para enfocarse en diferentes usuarios puede revelar patrones de consumo distintos. Algunos usuarios pueden tener hábitos de consumo más altos en ciertas fechas o con ciertos productos.

Plan de ejecución por Oracle/ Tiempos de ejecución:

PLAN_TABLE_OUTPUT

```
Plan hash value: 2513089040

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                         |     1 |   398 |     3   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                  |                         |     1 |   398 |     3   (0)| 00:00:01 |
|   2 |   NESTED LOOPS                 |                         |     1 |   398 |     3   (0)| 00:00:01 |
|   3 |    NESTED LOOPS                |                         |     1 |   258 |     3   (0)| 00:00:01 |
|   4 |     MERGE JOIN CARTESIAN       |                         |     1 |   223 |     3   (0)| 00:00:01 |
|   5 |      NESTED LOOPS              |                         |     1 |   179 |     1   (0)| 00:00:01 |
|   6 |       NESTED LOOPS             |                         |     1 |   179 |     1   (0)| 00:00:01 |
|   7 |        INDEX FULL SCAN         | SERVICIOS_PRODUCTOS_PK  |     1 |    26 |     1   (0)| 00:00:01 |
|*  8 |        INDEX UNIQUE SCAN       | PRODUCTOS_PK            |     1 |       |     0   (0)| 00:00:01 |
|   9 |       TABLE ACCESS BY INDEX ROWID| PRODUCTOS             |     1 |   153 |     0   (0)| 00:00:01 |
|  10 |      BUFFER SORT               |                         |     1 |    44 |     3   (0)| 00:00:01 |
|* 11 |       TABLE ACCESS FULL        | RESERVAS                |     1 |    44 |     2   (0)| 00:00:01 |
|  12 |     TABLE ACCESS BY INDEX ROWID| RESERVAS_SERVICIOS      |     1 |    35 |     0   (0)| 00:00:01 |
|* 13 |      INDEX UNIQUE SCAN         | RESERVAS_SERVICIOS_PK   |     1 |       |     0   (0)| 00:00:01 |
|* 14 |    INDEX UNIQUE SCAN           | USUARIOS_PK             |     1 |       |     0   (0)| 00:00:01 |
|* 15 |   TABLE ACCESS BY INDEX ROWID  | USUARIOS                |     1 |   140 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------
```

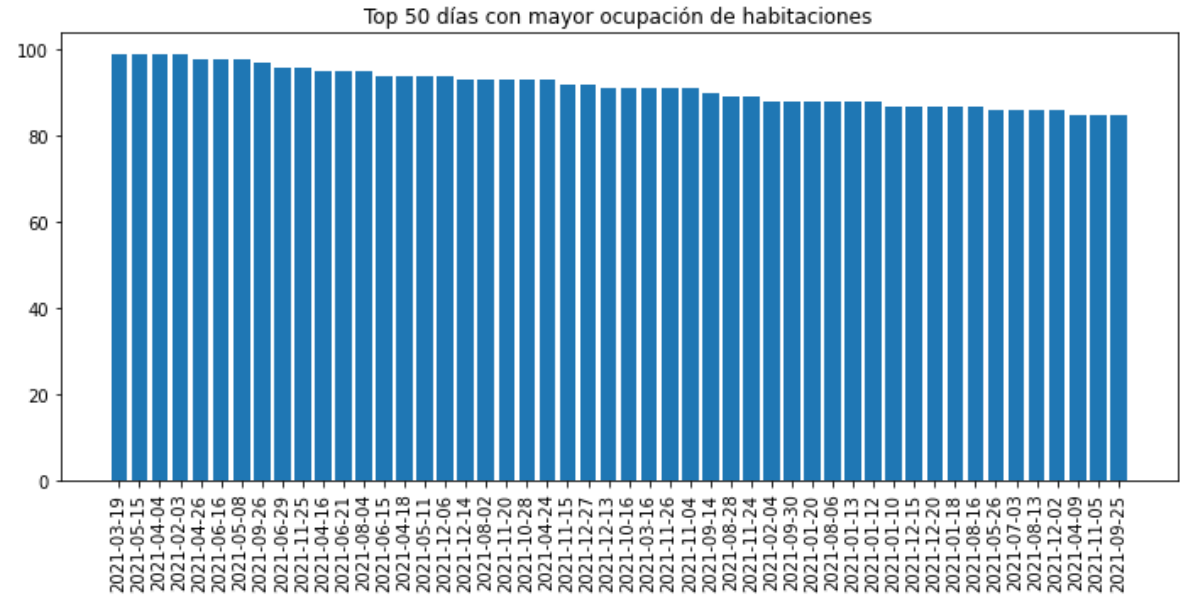
Como plan alternativo podría plantearse la creación de índices en reservas, para no realizar un TABLE ACCESS FULL. Así se lograría reducir el costo.

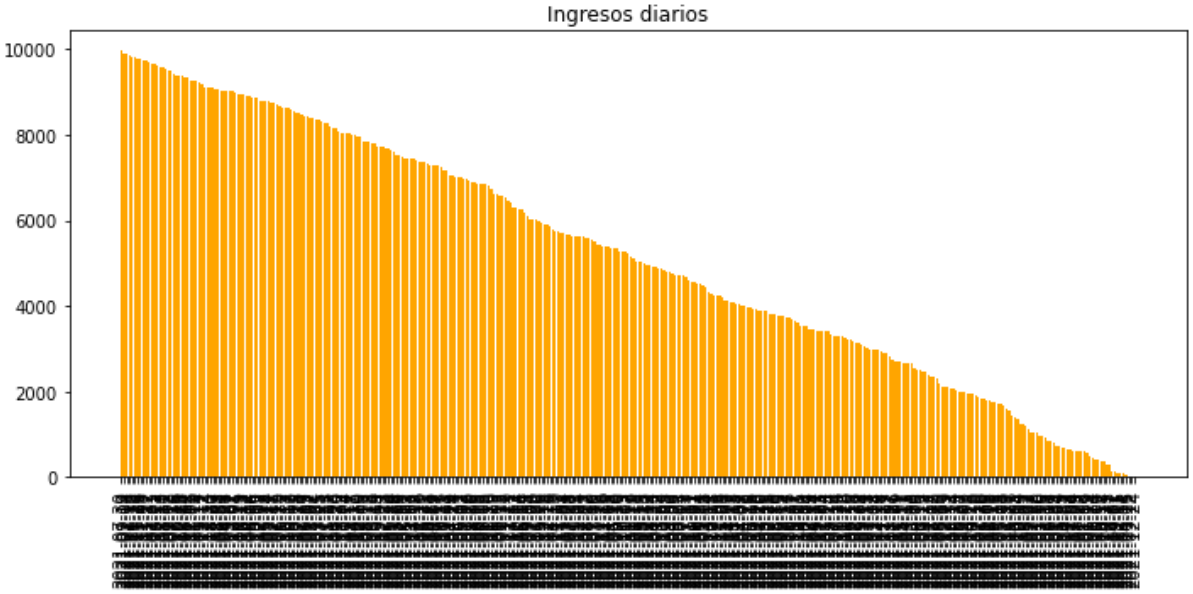
Requerimiento 6:
Para este requerimiento se realiza inicialmente una consulta llamada OcupacionDiaria para calcular la ocupación diaria de las habitaciones basándose en las reservas con check-in confirmado. Filtra las reservas por fechas entre las fechas mínimas y máximas encontradas en las entradas y salidas de todas las reservas. Luego, se agrupan los resultados por fecha_entrada y cuenta el número de habitaciones distintas ocupadas por día. Finalmente, selecciona las fechas y el número de habitaciones ocupadas, ordenando los resultados en orden descendente de habitaciones ocupadas.

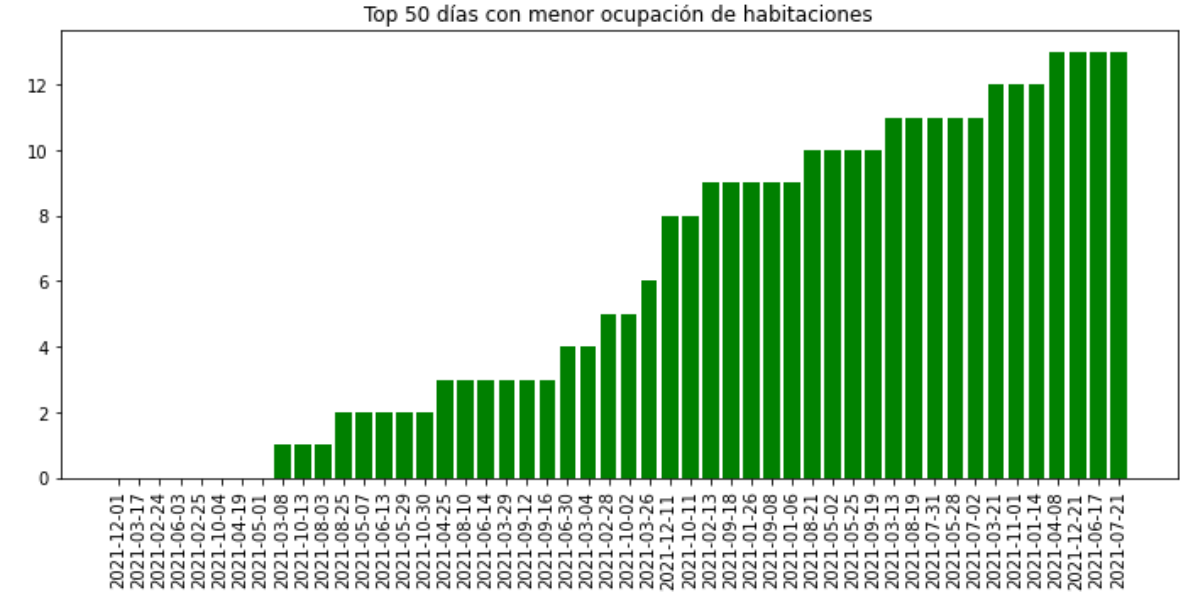
Este proceso se realiza de la misma manera para el cálculo de mayores ingresos, pero con se ordena con un orden ascendente.

```
@Query(value = "WITH OcupacionDiaria AS (\r\n" +
      "SELECT\r\n" +
         "fecha_entrada AS fecha_reserva,\r\n" +
         "COUNT(DISTINCT habitaciones_id) AS habitaciones_ocupadas\r\n" +
      "FROM\r\n" +
         "reservas\r\n" +
      "WHERE\r\n" +
         "check_in = 1 -- Asegúrate de que solo estás considerando reservas con check-in confirmado\r\n" +
         "AND fecha_entrada BETWEEN (SELECT MIN(fecha_entrada) FROM reservas) AND (SELECT
MAX(fecha_salida) FROM reservas)\r\n" +
      "GROUP BY\r\n" +
         "fecha_entrada\r\n" +
      ")\r\n" +
      "SELECT\r\n" +
         "fecha_reserva, habitaciones_ocupadas\r\n" +
      "FROM\r\n" +
         "OcupacionDiaria\r\n" +
      "ORDER BY\r\n" +
         "habitaciones_ocupadas DESC\r\n" +
      "FETCH FIRST 50 ROWS ONLY"
               , nativeQuery = true )
               Collection<Object[]> darPrimeraParte();



   @Query(value = "WITH IngresosDiarios AS (\r\n" +
      "SELECT\r\n" +
         "fecha_consumo AS fecha,\r\n" +
         "SUM(cont_servicios) AS total_ingresos\r\n" +
      "FROM\r\n" +
         "reservas_servicios\r\n" +
      "GROUP BY\r\n" +
         "fecha_consumo\r\n" +
   ")\r\n" +
   "SELECT\r\n" +
      "fecha, total_ingresos\r\n" +
   "FROM\r\n" +
```

```
        "IngresosDiarios\r\n" +
    "ORDER BY\r\n" +
        "total_ingresos DESC"
                , nativeQuery = true )
                Collection<Object[]> darSegundaParte();




    @Query(value = "WITH OcupacionDiaria AS (\r\n" +
        "SELECT\r\n" +
            "fecha_entrada AS fecha_reserva,\r\n" +
            "COUNT(DISTINCT habitaciones_id) AS habitaciones_ocupadas\r\n" +
        "FROM\r\n" +
            "reservas\r\n" +
        "WHERE\r\n" +
            "check_in = 1 -- Asegúrate de que solo estás considerando reservas con check-in confirmado\r\n" +
            "AND fecha_entrada BETWEEN (SELECT MIN(fecha_entrada) FROM reservas) AND (SELECT
MAX(fecha_salida) FROM reservas)\r\n" +
        "GROUP BY\r\n" +
            "fecha_entrada\r\n" +
        ")\r\n" +
        "SELECT\r\n" +
            "fecha_reserva, habitaciones_ocupadas\r\n" +
        "FROM\r\n" +
            "OcupacionDiaria\r\n" +
        "ORDER BY\r\n" +
            "habitaciones_ocupadas ASC\r\n" +
        "FETCH FIRST 50 ROWS ONLY"
                , nativeQuery = true )
                Collection<Object[]> darTerceraParte();
}
```


Top 50 días con mayor ocupación de habitaciones
100
80
60
40
20
0
2021-03-19
2021-05-15
2021-04-04
2021-02-03
2021-04-26
2021-06-16
2021-05-08
2021-09-26
2021-06-29
2021-11-25
2021-04-16
2021-06-21
2021-08-04
2021-06-15
2021-04-18
2021-05-11
2021-12-06
2021-12-14
2021-08-02
2021-11-20
2021-10-28
2021-04-24
2021-11-15
2021-12-27
2021-12-13
2021-10-16
2021-03-16
2021-11-26
2021-11-04
2021-09-14
2021-08-28
2021-11-24
2021-02-04
2021-09-30
2021-01-20
2021-08-06
2021-01-13
2021-01-12
2021-01-10
2021-12-15
2021-12-20
2021-01-18
2021-08-16
2021-05-26
2021-07-03
2021-08-13
2021-12-02
2021-04-09
2021-11-05
2021-09-25


Ingresos diarios
10000
8000
6000
4000
2000
0


Top 50 días con menor ocupación de habitaciones
12
10
8
6
4
2
0
2021-12-01
2021-03-17
2021-02-24
2021-06-03
2021-02-25
2021-10-04
2021-04-19
2021-05-01
2021-03-08
2021-10-13
2021-08-03
2021-08-25
2021-05-07
2021-06-13
2021-05-29
2021-10-30
2021-04-25
2021-08-10
2021-06-14
2021-03-29
2021-09-12
2021-09-16
2021-06-30
2021-03-04
2021-02-28
2021-10-02
2021-03-26
2021-12-11
2021-10-11
2021-02-13
2021-09-18
2021-01-26
2021-09-08
2021-01-06
2021-08-21
2021-05-02
2021-05-25
2021-09-19
2021-03-13
2021-08-19
2021-07-31
2021-05-28
2021-07-02
2021-03-21
2021-11-01
2021-01-14
2021-04-08
2021-12-21
2021-06-17
2021-07-21

Los gráficos reflejan cómo el tamaño de la respuesta (número de habitaciones ocupadas o ingresos diarios) puede cambiar según el valor de los parámetros utilizados (como fechas específicas).En el caso del gráfico de ocupación, si el período seleccionado fuera durante una temporada alta conocida, las barras serían consistentemente altas, lo que indica una respuesta grande (alta ocupación). En períodos de baja demanda, la respuesta sería más pequeña (menor ocupación).

Plan de ejecución por Oracle/ Tiempos de ejecución:

Primera parte:

```
Plan hash value: 2304871597

-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |           |    50 |  2400 |     9  (34)| 00:00:01 |
|   1 |  SORT ORDER BY                |           |    50 |  2400 |     9  (34)| 00:00:01 |
|*  2 |   VIEW                        |           |    50 |  2400 |     8  (25)| 00:00:01 |
|*  3 |    WINDOW SORT PUSHED RANK    |           |     1 |    22 |     8  (25)| 00:00:01 |
|   4 |     HASH GROUP BY             |           |     1 |    22 |     8  (25)| 00:00:01 |
|   5 |      VIEW                     | VM_NWVW_1 |     1 |    22 |     7  (15)| 00:00:01 |
|   6 |       HASH GROUP BY           |           |     1 |    25 |     7  (15)| 00:00:01 |
|*  7 |        TABLE ACCESS FULL      | RESERVAS  |     1 |    25 |     2   (0)| 00:00:01 |
|   8 |         SORT AGGREGATE        |           |     1 |     9 |            |          |
|   9 |          TABLE ACCESS FULL    | RESERVAS  |     1 |     9 |     2   (0)| 00:00:01 |
|  10 |         SORT AGGREGATE        |           |     1 |     9 |            |          |
|  11 |          TABLE ACCESS FULL    | RESERVAS  |     1 |     9 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

En este caso, como plan alternativo podría plantearse la creación de un índice para reservas, para que no sea necesario realizar un TABLE ACCESS FULL, y así poder reducir el costo de la consulta.

Segunda parte:

```
Plan hash value: 2505605778

-----------------------------------------------------------------------------------------------
| Id  | Operation           | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                     |     1 |    22 |     4  (50)| 00:00:01 |
|   1 |  SORT ORDER BY      |                     |     1 |    22 |     4  (50)| 00:00:01 |
|   2 |   HASH GROUP BY     |                     |     1 |    22 |     4  (50)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| RESERVAS_SERVICIOS  |     1 |    22 |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

En este caso, como plan alternativo podría plantearse la creación de un índice para reservas_servicios, para que no sea necesario realizar un TABLE ACCESS FULL, y así poder reducir el costo de la consulta.

Tercera parte:

```
Plan hash value: 2304871597

-------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |           |    50 |  2400 |     9  (34)| 00:00:01 |
|   1 |  SORT ORDER BY             |           |    50 |  2400 |     9  (34)| 00:00:01 |
|*  2 |   VIEW                     |           |    50 |  2400 |     8  (25)| 00:00:01 |
|*  3 |    WINDOW SORT PUSHED RANK |           |     1 |    22 |     8  (25)| 00:00:01 |
|   4 |     HASH GROUP BY          |           |     1 |    22 |     8  (25)| 00:00:01 |
|   5 |      VIEW                  | VM_NWVW_1 |     1 |    22 |     7  (15)| 00:00:01 |
|   6 |       HASH GROUP BY        |           |     1 |    25 |     7  (15)| 00:00:01 |
|*  7 |        TABLE ACCESS FULL   | RESERVAS  |     1 |    25 |     2   (0)| 00:00:01 |
|   8 |         SORT AGGREGATE     |           |     1 |     9 |            |          |
|   9 |          TABLE ACCESS FULL | RESERVAS  |     1 |     9 |     2   (0)| 00:00:01 |
|  10 |         SORT AGGREGATE     |           |     1 |     9 |            |          |
|  11 |          TABLE ACCESS FULL | RESERVAS  |     1 |     9 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

En este caso, como plan alternativo podría plantearse la creación de un índice para reservas, para que no sea necesario realizar un TABLE ACCESS FULL, y así poder reducir el costo de la consulta.

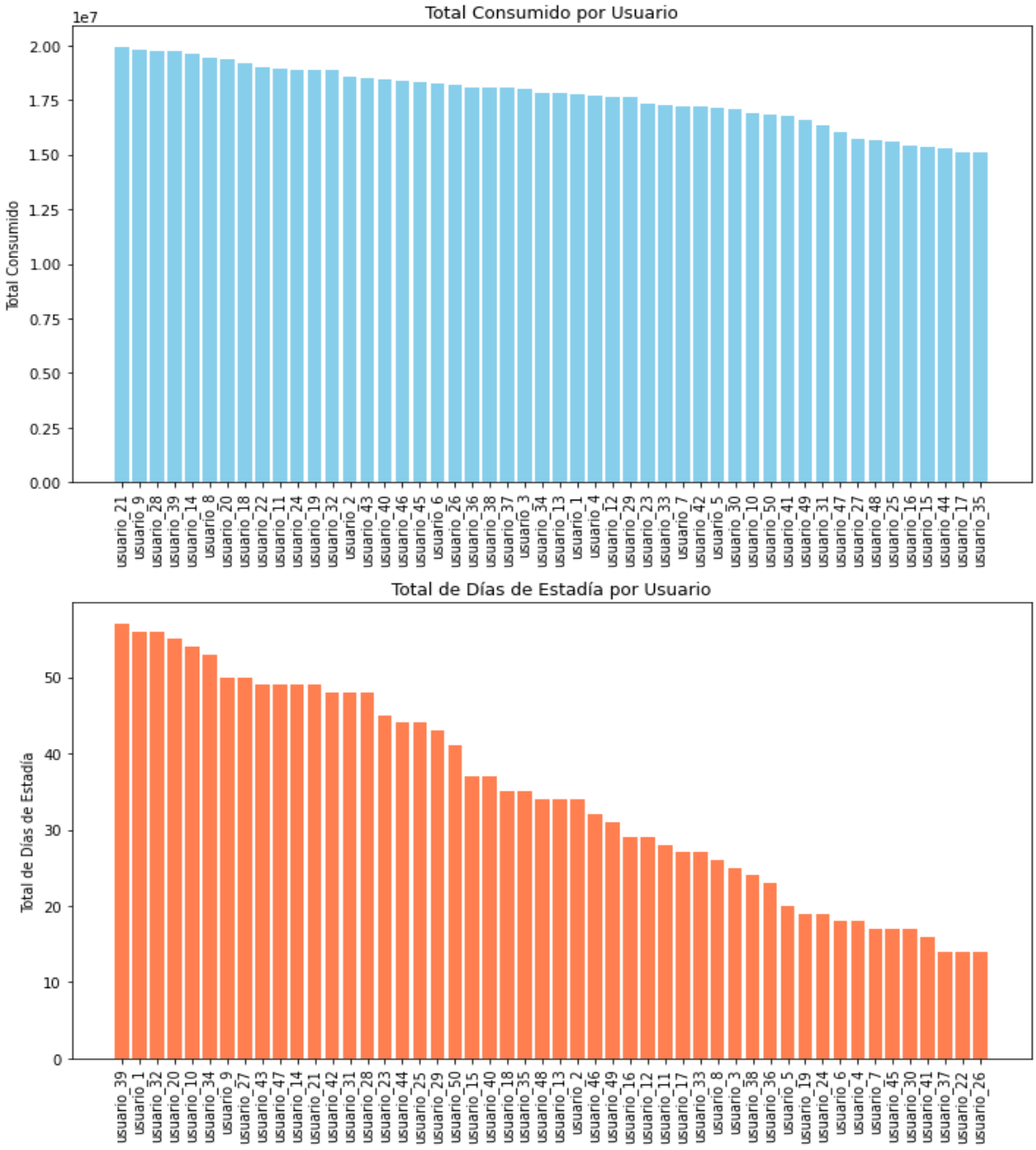
Requerimiento 7:

Para este requerimiento, primeramente calculamos el consumo total de cada usuario (total_consumido) en reservas durante el último año (hasta la fecha actual), y se revisa si el resultado es mayor a $15.000.000. Luego se hace otra subconsulta que calcula el total de días de estadía (total_dias_estadia) de cada usuario, donde el total de días es igual o superior a 14 días. Luego se filtran las reservas que tienen una fecha_entrada dentro del último año y hasta la fecha actual y se agrupan los resultados por usuario_id y usuario_login.

Finalmente se realiza un join entre las dos subconsultas en el campo ‘usuario_id’ y Devuelve una colección de objetos donde cada objeto contiene los campos usuario_id, usuario_login,

total_dias_estadia, y total_consumido para los usuarios que cumplen con ambas condiciones: un consumo total mayor a $15.000.000 y una estadía total igual o mayor a 14 días, en el último año.

```
@Query(value = "SELECT consumo.usuario_id, consumo.usuario_login, estadia.total_dias_estadia,
total_consumido\r\n" + //
        "FROM (\r\n" +
        "    -- Resultado del cálculo de consumo\r\n" +
        "    SELECT u.id AS usuario_id, u.login AS usuario_login, SUM(r.costo_total) AS total_consumido\r\n" +
        "    FROM usuarios u\r\n" +
        "    JOIN reservas r ON u.id = r.usuarios_id\r\n" +
        "    WHERE r.fecha_entrada >= TO_DATE(SYSDATE - 365, 'DD/MM/YYYY')\r\n" +
        "    GROUP BY u.id, u.login\r\n" +
        "    HAVING SUM(r.costo_total) > 15000000\r\n" +
        ") consumo\r\n" +
        "JOIN (\r\n" +
        "    -- Resultado del cálculo de duración de estadía\r\n" +
        "    SELECT u.id AS usuario_id, u.login AS usuario_login, SUM(ABS(r.fecha_salida - r.fecha_entrada)) AS
total_dias_estadia\r\n" +
        "    FROM usuarios u\r\n" +
        "    JOIN reservas r ON u.id = r.usuarios_id\r\n" +
        "    WHERE r.fecha_entrada >= TO_DATE(SYSDATE - 365, 'DD/MM/YYYY') and r.fecha_entrada <=
(SELECT CURRENT_DATE from dual)\r\n" +
        "    GROUP BY u.id, u.login\r\n" +
        "    HAVING SUM(ABS(r.fecha_salida - r.fecha_entrada)) >= 14\r\n" +
        ") estadia\r\n" +
        "ON consumo.usuario_id = estadia.usuario_id\r\n" +
        "", nativeQuery = true )
        Collection<Object[]> darRta();
}
```



Cambiar los parámetros de la consulta para enfocarse en periodos de tiempo específicos podría alterar significativamente el tamaño de la respuesta. Por ejemplo, incluir solo días de temporada alta probablemente mostraría ingresos consistentemente altos, mientras que incluir una temporada baja podría mostrar una mayor variabilidad y disminuir el tamaño de la respuesta. El tamaño de la respuesta está directamente relacionado con la selectividad de los parámetros de la consulta. Parámetros más restrictivos harán que la respuesta sea más selectiva y posiblemente más pequeña, mientras que parámetros más amplios podrían capturar una gama más amplia de comportamientos y aumentar el tamaño de la respuesta.

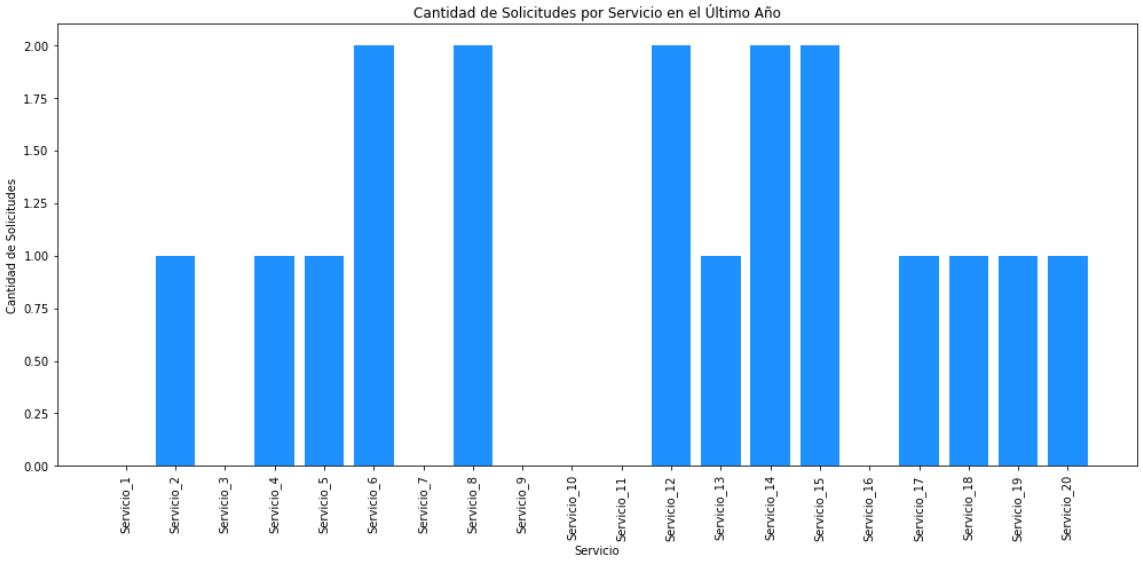
Plan de ejecución por Oracle/ Tiempos de ejecución:

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	179	8 (25)	00:00:01
* 1	HASH JOIN		1	179	8 (25)	00:00:01
2	VIEW		1	153	3 (34)	00:00:01
* 3	FILTER					
4	HASH GROUP BY		1	175	3 (34)	00:00:01
5	NESTED LOOPS		1	175	2 (0)	00:00:01
6	NESTED LOOPS		1	175	2 (0)	00:00:01
* 7	TABLE ACCESS FULL	RESERVAS	1	35	2 (0)	00:00:01
* 8	INDEX UNIQUE SCAN	USUARIOS_PK	1		0 (0)	00:00:01
9	TABLE ACCESS BY INDEX ROWID	USUARIOS	1	140	0 (0)	00:00:01
10	VIEW		1	26	5 (20)	00:00:01
* 11	FILTER					
12	HASH GROUP BY		1	171	5 (20)	00:00:01
* 13	FILTER					
14	NESTED LOOPS		1	171	2 (0)	00:00:01
15	NESTED LOOPS		1	171	2 (0)	00:00:01
* 16	TABLE ACCESS FULL	RESERVAS	1	31	2 (0)	00:00:01
17	FAST DUAL		1		2 (0)	00:00:01
* 18	INDEX UNIQUE SCAN	USUARIOS_PK	1		0 (0)	00:00:01
19	TABLE ACCESS BY INDEX ROWID	USUARIOS	1	140	0 (0)	00:00:01

En este caso, como plan alternativo podría plantearse la creación de un índice para reservas, para que no sea necesario realizar un TABLE ACCESS FULL, y así poder reducir el costo de la consulta.

Requerimiento 8:
Para este requerimiento se unen las tablas ‘reservas_servicios’ con ‘servicios’ y ‘reservas’ con ‘reservas_servicios’. Donde se tienen en cuenta los datos para el último año de operación, se agrupan los datos por nombre y filtra los datos para obtener aquellos servicios que han sido menos solicitados.

```
@Query(value = "SELECT s.nombre AS nombre_del_servicio, COUNT(rs.reservas_id) AS
cantidad_de_solicitudes\r\n" +
        "FROM servicios s\r\n" +
        "LEFT JOIN reservas_servicios rs ON s.id = rs.servicios_id\r\n" +
        "LEFT JOIN reservas r ON rs.reservas_id = r.id\r\n" +
        "WHERE r.fecha_entrada >= (SYSDATE - 365) -- Último año de operación\r\n" +
        "GROUP BY s.nombre\r\n" +
        "HAVING COUNT(rs.reservas_id) < 3\r\n" +
        "ORDER BY cantidad_de_solicitudes\r\n" +
        "", nativeQuery = true )
   Collection<Object[]> darRta();
}
```



Si el rango de fechas incluye un periodo de tiempo durante el cual el hotel ha estado completamente operativo y sin cambios significativos en la oferta de servicios, es posible que menos servicios cumplan con el criterio de menos de tres solicitudes, reduciendo el tamaño de la respuesta. El tamaño de la respuesta está directamente relacionado con la selectividad de los parámetros de la consulta. Ajustar el umbral de solicitudes de "menos de tres" a "menos de cinco" podría aumentar el tamaño de la respuesta, mientras que aumentar el umbral a "menos de dos" podría disminuirlo.

Plan de ejecución por Oracle/ Tiempos de ejecución:

```
-------------------------------------------------------------------------------------------
| Id  | Operation                      | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |              |     1 |   179 |     8  (25)| 00:00:01 |
|*  1 |  HASH JOIN                     |              |     1 |   179 |     8  (25)| 00:00:01 |
|   2 |   VIEW                         |              |     1 |   153 |     3  (34)| 00:00:01 |
|*  3 |    FILTER                      |              |       |       |            |          |
|   4 |     HASH GROUP BY              |              |     1 |   175 |     3  (34)| 00:00:01 |
|   5 |      NESTED LOOPS              |              |     1 |   175 |     2   (0)| 00:00:01 |
|   6 |       NESTED LOOPS             |              |     1 |   175 |     2   (0)| 00:00:01 |
|*  7 |        TABLE ACCESS FULL       | RESERVAS     |     1 |    35 |     2   (0)| 00:00:01 |
|*  8 |        INDEX UNIQUE SCAN       | USUARIOS_PK  |     1 |       |     0   (0)| 00:00:01 |
|   9 |       TABLE ACCESS BY INDEX ROWID | USUARIOS  |     1 |   140 |     0   (0)| 00:00:01 |
|  10 |   VIEW                         |              |     1 |    26 |     5  (20)| 00:00:01 |
|* 11 |    FILTER                      |              |       |       |            |          |
|  12 |     HASH GROUP BY              |              |     1 |   171 |     5  (20)| 00:00:01 |
|* 13 |      FILTER                    |              |       |       |            |          |
|  14 |       NESTED LOOPS             |              |     1 |   171 |     2   (0)| 00:00:01 |
|  15 |        NESTED LOOPS            |              |     1 |   171 |     2   (0)| 00:00:01 |
|* 16 |         TABLE ACCESS FULL      | RESERVAS     |     1 |    31 |     2   (0)| 00:00:01 |
|  17 |          FAST DUAL             |              |     1 |       |     2   (0)| 00:00:01 |
|* 18 |         INDEX UNIQUE SCAN      | USUARIOS_PK  |     1 |       |     0   (0)| 00:00:01 |
|  19 |        TABLE ACCESS BY INDEX ROWID| USUARIOS  |     1 |   140 |     0   (0)| 00:00:01 |
```

En este caso, como plan alternativo podría plantearse la creación de un índice para reservas, para que no sea necesario realizar un TABLE ACCESS FULL, y así poder reducir el costo de la consulta.

Los valores utilizados en las gráficas son valores aleatorios, usados a método ilustrativo del funcionamiento de los requerimientos en diversos datos.

Los valores utilizados para probar los requerimientos, fueron basados en las siguientes tablas:

```
CREATE TABLE carachabitaciones (
   nombre VARCHAR2(250) NOT NULL,
   id     NUMBER NOT NULL
);

ALTER TABLE carachabitaciones ADD CONSTRAINT carachabitaciones_pk PRIMARY KEY ( id );

CREATE TABLE caract_habitacion (
   carachabitaciones_id NUMBER NOT NULL,
   habitaciones_id      NUMBER NOT NULL
);

ALTER TABLE caract_habitacion ADD CONSTRAINT caract_habitacion_pk PRIMARY KEY (
carachabitaciones_id,
                                                habitaciones_id );

CREATE TABLE habitaciones (
   costo_noche     NUMBER NOT NULL,
   tipo_habitacion VARCHAR2(250) NOT NULL,
   id              NUMBER NOT NULL,
   hoteles_id      NUMBER NOT NULL,
   cont_habitaciones number
);

ALTER TABLE habitaciones ADD CONSTRAINT habitaciones_pk PRIMARY KEY ( id );

CREATE TABLE hoteles (
   nombre    VARCHAR2(250) NOT NULL,
   tipo      VARCHAR2(250) NOT NULL,
   estrellas NUMBER NOT NULL,
   id        NUMBER NOT NULL
);

ALTER TABLE hoteles ADD CONSTRAINT hoteles_pk PRIMARY KEY ( id );

CREATE TABLE planes_de_cosumo (
   comida CHAR(1) NOT NULL,
   bebida CHAR(1) NOT NULL,
   id     NUMBER NOT NULL,
   nombre VARCHAR2(250) NOT NULL
);

ALTER TABLE planes_de_cosumo ADD CONSTRAINT planes_de_cosumo_pk PRIMARY KEY (
id );
```

```
CREATE TABLE productos (
   nombre VARCHAR2(250) NOT NULL,
   id     NUMBER NOT NULL,
   costo  NUMBER NOT NULL,
   tipo VARCHAR2(250) NOT NULL
);

ALTER TABLE productos ADD CONSTRAINT productos_pk PRIMARY KEY ( id );

CREATE TABLE reservas (
   numero_personas     NUMBER NOT NULL,
   id                  NUMBER NOT NULL,
   fecha_entrada       DATE NOT NULL,
   fecha_salida        DATE NOT NULL,
   costo_total         NUMBER NOT NULL,
   check_in            CHAR(1) NOT NULL,
   check_out           CHAR(1) NOT NULL,
   planes_de_cosumo_id NUMBER,
   hoteles_id          NUMBER NOT NULL,
   Usuarios_id         NUMBER NOT NULL,
   habitaciones_id     NUMBER NOT NULL
);

ALTER TABLE reservas ADD CONSTRAINT reservas_pk PRIMARY KEY ( id );

CREATE TABLE reservas_servicios (
   reservas_id  NUMBER NOT NULL,
   servicios_id NUMBER NOT NULL,
   fecha_consumo DATE NOT NULL,
   cont_servicios number not null
);

ALTER TABLE reservas_servicios ADD CONSTRAINT reservas_servicios_pk PRIMARY KEY (
reservas_id,
                                                    servicios_id );

CREATE TABLE servicios (
   nombre VARCHAR2(250) NOT NULL,
   id     NUMBER NOT NULL,
   tipo_servicio VARCHAR2(250),
   precio NUMBER NOT NULL
);

ALTER TABLE servicios ADD CONSTRAINT servicios_pk PRIMARY KEY ( id );

CREATE TABLE servicios_productos (
```

```
   servicios_id NUMBER NOT NULL,
   productos_id NUMBER NOT NULL
);

ALTER TABLE servicios_productos ADD CONSTRAINT servicios_productos_pk PRIMARY KEY (
servicios_id,
                                                productos_id );

CREATE TABLE usuarios (
   id          NUMBER NOT NULL,
   login       VARCHAR2(250) NOT NULL,
   contraseña   VARCHAR2(250) NOT NULL,
   nombre      VARCHAR2(250) NOT NULL,
   no_documento NUMBER NOT NULL,
   tipo_usuario VARCHAR2(250) NOT NULL,
   reservas_id  NUMBER NOT NULL,
   hoteles_id   NUMBER NOT NULL
);

CREATE UNIQUE INDEX usuarios__idx ON
   usuarios (
      reservas_id
   ASC );

ALTER TABLE usuarios ADD CONSTRAINT usuarios_pk PRIMARY KEY ( id );

--  ERROR: FK name length exceeds maximum allowed length(30)
ALTER TABLE caract_habitacion
   ADD CONSTRAINT caracthabitacion_fk FOREIGN KEY ( carachabitaciones_id )
      REFERENCES carachabitaciones ( id );

--  ERROR: FK name length exceeds maximum allowed length(30)
ALTER TABLE caract_habitacion
   ADD CONSTRAINT caract_habitacion_fk FOREIGN KEY ( habitaciones_id )
      REFERENCES habitaciones ( id );

ALTER TABLE habitaciones
   ADD CONSTRAINT habitaciones_hoteles_fk FOREIGN KEY ( hoteles_id )
      REFERENCES hoteles ( id );

ALTER TABLE reservas
   ADD CONSTRAINT reservas_hoteles_fk FOREIGN KEY ( hoteles_id )
      REFERENCES hoteles ( id );

ALTER TABLE reservas
   ADD CONSTRAINT reservas_planes_de_cosumo_fk FOREIGN KEY ( planes_de_cosumo_id )
      REFERENCES planes_de_cosumo ( id );
```

```
ALTER TABLE reservas_servicios
   ADD CONSTRAINT reservasservicios_fk FOREIGN KEY ( reservas_id )
      REFERENCES reservas ( id );

--  ERROR: FK name length exceeds maximum allowed length(30)
ALTER TABLE reservas_servicios
   ADD CONSTRAINT reservas_servicios_fk FOREIGN KEY ( servicios_id )
      REFERENCES servicios ( id );

--  ERROR: FK name length exceeds maximum allowed length(30)
ALTER TABLE servicios_productos
   ADD CONSTRAINT serviciosproductos_fk FOREIGN KEY ( productos_id )
      REFERENCES productos ( id );

--  ERROR: FK name length exceeds maximum allowed length(30)
ALTER TABLE servicios_productos
   ADD CONSTRAINT servicios_productos_fk FOREIGN KEY ( servicios_id )
      REFERENCES servicios ( id );

ALTER TABLE usuarios
   ADD CONSTRAINT usuarios_hoteles_fk FOREIGN KEY ( hoteles_id )
      REFERENCES hoteles ( id );

ALTER TABLE reservas
   ADD CONSTRAINT reservas_usuarios_fk FOREIGN KEY ( usuarios_id )
      REFERENCES usuarios ( id );

ALTER TABLE reservas
   ADD CONSTRAINT reservas_habitaciones_fk FOREIGN KEY ( habitaciones_id )
      REFERENCES habitaciones ( id );


commit;
```