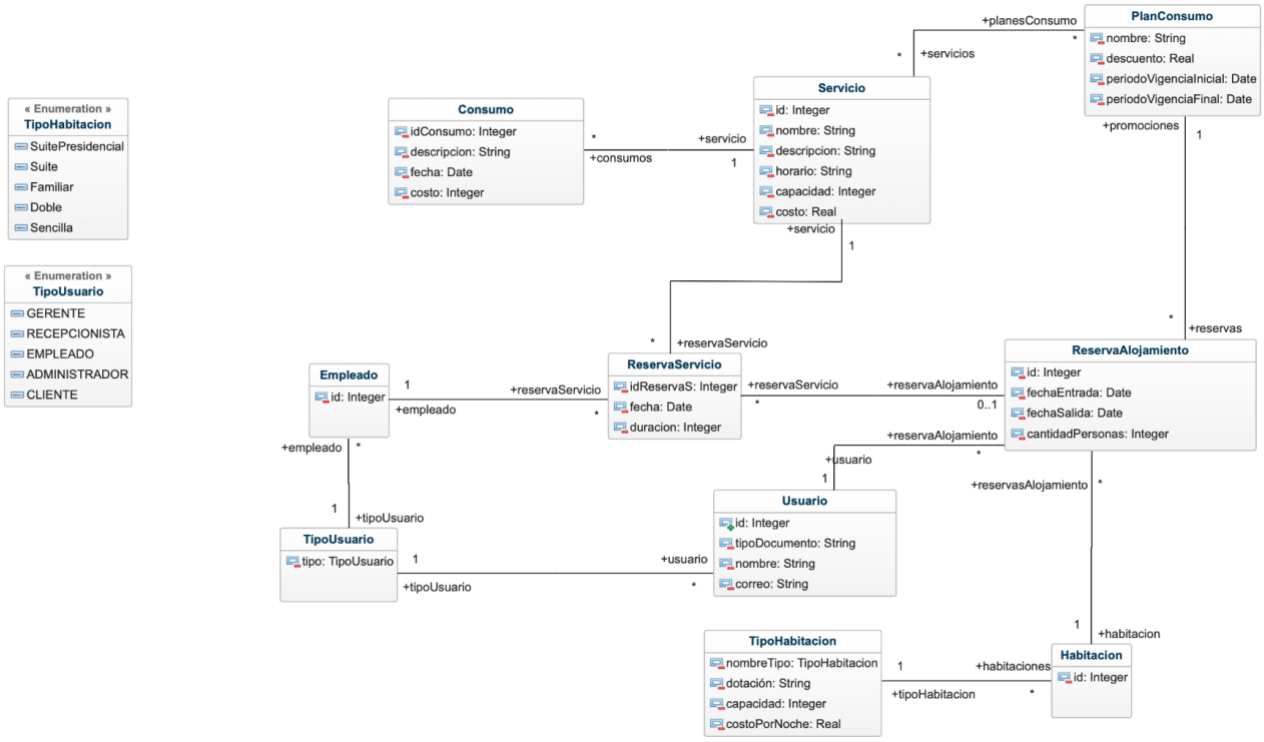
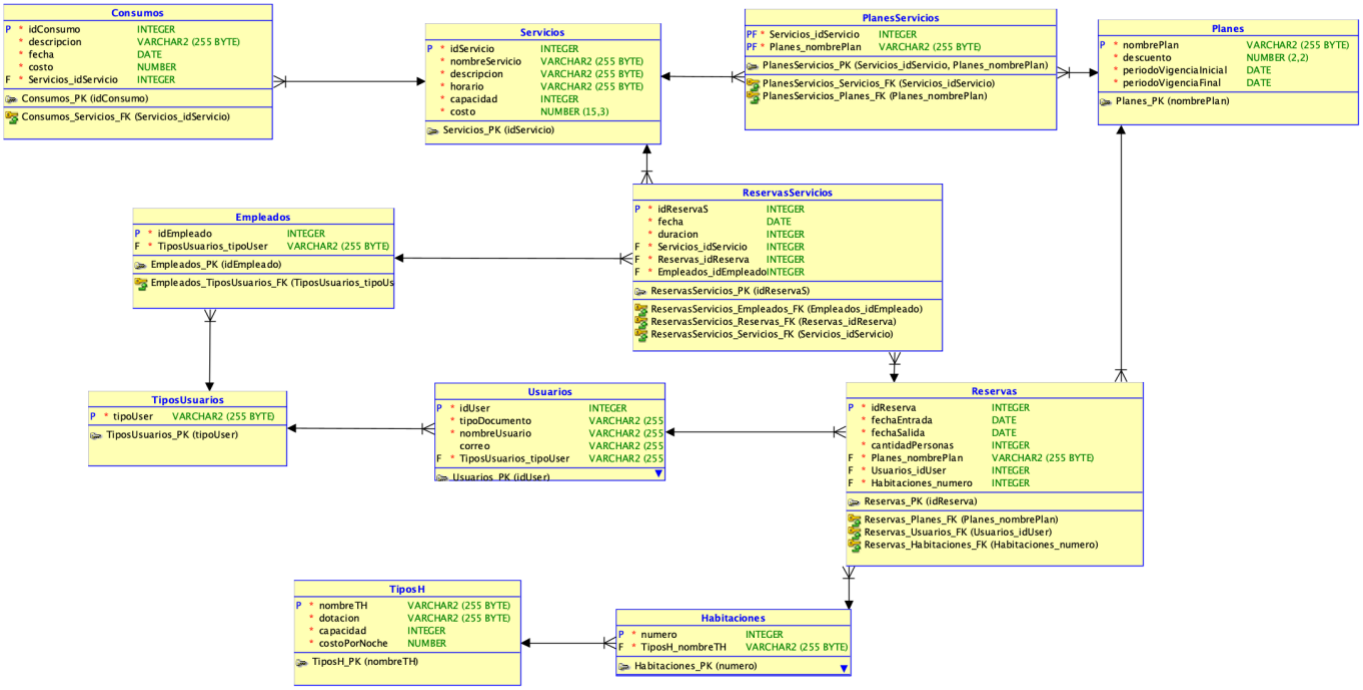
INFORME P2 SISTEMAS TRANSACCIONALES

MODELOS

UML



E/R



CARGA DE DATOS

Para la carga de datos se escogió la metodología de hacer un script en Python el cual generara las sentencias SQL que contenían los datos que se iban a meter dentro de la base de datos de Oracle. El script de Python se escogió debido a que podemos tener control sobre los datos que generamos y la consistencia entre los datos que generamos, esto debido a que hay varias reglas de negocio que deben seguir los datos al entrar dentro de la base de datos.

El proceso para generar los datos empieza por crear los datos en tablas que no dependen de otras tablas, es decir, que no tienen FKs. Por ejemplo, se crearon los datos para la tabla de “TipoUsuarios”, “TiposH”, “Planes” y “Servicios”. Estas tablas no dependen de ninguna otra tabla por lo cual podemos crear los datos iniciales de forma manual y meterlos dentro de la base de datos. De aquí en adelante, la información que se creó para el resto de las tablas fue de manera pseudoaleatorio. Por ejemplo, se creó un numero arbitrario de habitaciones escogiendo aleatoriamente que tipo de habitación iban a hacer asignándoles una llave aleatoria que estuviera en “TiposH”. Para empleados y usuarios se siguió la misma metodología, pero ateniéndonos a las reglas de negocios como que un usuario solo puede ser cliente mientras un empleado puede tener otro rol dentro del hotel. Para la tabla de “PlanesServicios” se asignaron los servicios correspondientes al plan y algunos más.

Por otro lado, para crear las reservas del hotel, se cogieron llaves primarias de las tablas de “Habitaciones”, “Usuarios” y “Planes” para crear aleatoriamente datos sobre reservas hechas en el hotel, pero se debieron seguir algunas restricciones. Fue necesario asegurarse de que la fecha de salida registrada a una reserva fuera mayor a su fecha de entrada para evitar

inconsistencias lógicas dentro del sistema. Además, fue necesario asegurarnos de que una habitación no tuviera dos reservas que sobrelapen en la estadía. Esto igual se verifica en la aplicación, pero cuando se meten datos generados pseudoaleatoriamente es necesario revisar porque los datos no se están metiendo por la aplicación y para evitar errores en la base de datos. Para tablas como “Consumos” y “ReservasServicios” se siguió un proceso similar al de la aleatoriedad, pero de igual manera fue necesario implementar algunos chequeos. Fue necesario revisar que una reserva de servicio asignada a una reserva dentro del hotel estuviera hecha dentro de las fechas de estadía establecidas por la reserva, esto con el fin de que la información dentro de la base de datos fuera consistente.

El proceso de generación de datos con Python facilito la generación de las sentencias SQL para después ponerlas a correr en batch para subirlas a la base de datos. Para garantizar la consistencia de los datos, como por ejemplo que los datos que tuvieras FKs tuvieran una FK valida, localmente se guardaron las PKs de las tablas necesarias para poder asignar a las tuplas que lo necesitaran valores existentes sin necesidad de pedírselo a la base de datos. Además de las PKs, también se guardó otra información relevante que facilito la creación de datos. La decisión de guardar información localmente se hizo con el propósito de evitar peticiones a la base de datos mientras se crean los datos. Aunque las peticiones a la base de datos no tomen más de unos segundos por mucho, para la creación de casi un millón de datos el tiempo necesario hubiera sido mayor a un día por lo cual se decidió guardar toda la información localmente y después subirla toda en batch a la base de datos.

Cantidad de datos:

Tipos de habitaciones = 5
Tipos de usuarios = 5
Servicios = 23
Planes = 4
Servicios en planes = 40
Habitaciones = 10,000
Empleados = 50,000
Usuarios = 120,000
Consumos = 100,000
Reservas = 300,000
Reservas de servicios = 200,000

Total = 780,077 tuplas en la base de datos

Fragmentos de código

En el siguiente fragmento de código se evidencia como a cada una de las habitaciones se les asigna un tipo de habitación aleatoriamente de las PKs de tipo de habitación y se guarda su sentencia para subir a la base de datos.

```python
# Habitaciones

habitaciones_pk = range(0, num_habitaciones)

with open("habitaciones.txt", "w") as file:
    for id_habitacion in habitaciones_pk:
        file.write(f"INSERT INTO habitaciones VALUES ({id_habitacion}, '{random.choice(tipos_habitacion_pk)}');\n")
```

En el siguiente fragmento de código se evidencia como a cada usuario (tabla “Usuarios”) solo se le puede asignar el rol de “Cliente” y además que su tipo de documento solo puede ser CC, TI o PASAPORTE. Subsecuentemente se procede a guardar la sentencia con sus respectivos datos.

```python
# Usuarios

usuarios_pk = range(0, num_usuarios)

with open("usuarios.txt", "w") as file:
    tipo_documento = ["CC", "PASAPORTE", "TI"]
    tipo_usuario = "Cliente"
    for id_usuario in usuarios_pk:
        name = random.choice(names)
        file.write(f"INSERT INTO usuarios VALUES ({id_usuario}, '{random.choice(tipo_documento)}', '{name}', '{name.lower()}@gmail.com', '{tipo_usuario}');\n")
```

En el siguiente fragmento de código se puede ver como se crean datos para una fecha de entrada y una fecha de salida que se asignara a una reserva. El código revisa que la fecha de salida no sea menor al de entrada ya que estoy no tiene sentido lógico con las reglas de negocio. Si se determina que la fecha de salida es mayor que la fecha de entrada el código sale del while y sigue con sus tareas. En caso de que la fecha de salida sea menor que la fecha de entrada entonces se repite el proceso hasta que ya no sea el caso.

```python
while True:
    entry_year = random.randint(2020, 2023)
    entry_month = random.randint(1, 12)
    entry_day = random.randint(1, 28)

    exit_year = random.randint(entry_year, entry_year + 2)
    exit_month = random.randint(1, 12)
    exit_day = random.randint(1, 28)

    if (exit_year < entry_year):
        continue

    if (exit_year == entry_year and exit_month < entry_month):
        continue

    if (exit_year == entry_year and exit_month == entry_month and exit_day < entry_day):
        continue

    break
```

En el siguiente fragmento de código se puede ver como se asegura que una reserva de un servicio hecha asignada a una reserva este dentro de las fechas de entrada y de salida de la reserva. Para lograr esto se coge un día aleatorio entre las fechas de entrada y salida de la reserva y se le asigna a la reserva de servicio.

```
with open("reservas_servicios.txt", "w") as file:
    for id_reserva_servicio in reservas_servicios_pk:
        reserva = random.choice(reservas_pk)
        rango_fecha = reservas_fechas[reserva]
        fecha_entrada = rango_fecha[0]
        fecha_salida = rango_fecha[1]

        diferencia = (fecha_salida[0] - fecha_entrada[0]) * 365 + (fecha_salida[1] - fecha_entrada[1]) * 30 + (fecha_salida[2] - fecha_entrada[2])

        servicio_dias = random.randint(0, diferencia)

        extra_years = servicio_dias // 365
        extra_months = (servicio_dias % 365) // 30
        extra_days = (servicio_dias % 365) % 30

        fecha_servicio = (fecha_entrada[0] + extra_years, fecha_entrada[1] + extra_months, fecha_entrada[2] + extra_days)

        if (fecha_servicio[2] > 28):
            fecha_servicio = (fecha_servicio[0], fecha_servicio[1] + 1, fecha_servicio[2] % 28 if fecha_servicio[2] % 28 > 0 else 1)

        if (fecha_servicio[1] > 12):
            fecha_servicio = (fecha_servicio[0] + 1, fecha_servicio[1] % 12 if fecha_servicio[1] % 12 > 0 else 1, fecha_servicio[2])

        fecha = f"TO_DATE('{fecha_servicio[0]}-{fecha_servicio[1]}-{fecha_servicio[2]}', 'YYYY-MM-DD')"
        servicio = random.choice(servicios_pk)
        empleado = random.choice(empleados_pk)
        file.write(f"INSERT INTO reservasservicios VALUES ({id_reserva_servicio}, {fecha}, {random.randint(1, 7)}, {servicio}, {reserva}, {empleado});\n")
```

ÍNDICES

1. **Justificación:** Para el RFC1 no consideramos pertinente la creación de un índice porque la consulta que cumple con este RFC hace uso de funciones agregadas para hacer la suma del dinero recolectado sobre el atributo costo y simultáneamente necesita filtrar el atributo año en el campo WHERE. Por este motivo, aunque un índice puede acelerar la consulta en algunos casos de uso de función agregada, la utilización de criterios en el WHERE diferentes a los que se están usando en las funciones agregadas hace que el índice sea muy costoso de utilizar y eventualmente no valga la pena crearlo porque la reducción del tiempo de consulta no representa un valor significativo o justifica el mantenimiento del índice.
   Tipo de índice utilizado: ninguno

2. **Justificación:** Para el RFC2 no consideramos pertinente la creación de un índice porque con el volumen de datos que se va a manejar es muy probable que la selectividad sea baja para los registros que se van a operar. Si por ejemplo el atributo idServicio tiene 10000 registros y los 20 tipos de servicio se reparten equitativamente en estos registros (el cual es el mejor caso de selectividad), la selectividad sería de 500/10000 o el 5%. Dado lo anterior, no se va a crear un índice porque este no va a reducir significativamente el número de registros que se deben revisar para cumplir con la consulta.
   Tipo de índice utilizado: ninguno

3. **Justificación:** Para el RFC3 no consideramos pertinente la creación de un índice porque la consulta que cumple con este RFC hace uso de funciones agregadas para sumar los días de ocupación y simultáneamente filtra en el WHERE que el año si corresponda al último año por calendario. Por este motivo, aunque un índice puede acelerar la consulta en algunos casos de uso de función agregada, la utilización de criterios en el WHERE diferentes a los que se están usando en las funciones agregadas hace que el índice sea muy costoso de utilizar y eventualmente no valga la pena crearlo porque la reducción del tiempo de consulta no representa un valor significativo o justifica el mantenimiento del índice.
   Tipo de índice utilizado: ninguno

4. **Justificación:** en el RFC4 creemos que es pertinente la creación de múltiples índices para la optimización de las consultas que responden a este requerimiento. En primer lugar, se va a crear un índice sobre el atributo costo (tabla SERVICIOS) y otro sobre el atributo fecha (tabla RESERVASSERVICIO) ya que ambas consultas hacen una búsqueda por rangos en criterios de alta selectividad, por lo que se va a ver una reducción eficiente de los tiempos de consulta. Sin embargo, no se van a crear índices para agilizar la búsqueda de los registros que fueros registrados por cierto empleado o que son de cierto tipo porque los resultados de estas consultas en particular tienen una selectividad muy baja ya que los pocos servicios disponibles y los pocos empleados se van a repetir en múltiples ocasiones a lo largo de los registros.
   Tipo de índice utilizado: árbol B+ por su eficiencia en la búsqueda de rangos

5. **Justificación:** Para el RFC5 no consideramos pertinente la creación de un índice porque la consulta que cumple con este RFC hace uso de funciones agregadas para sumar el total del gasto del cliente en el hotel y, simultáneamente, necesita filtrar el criterio de año en el campo WHERE. Como la utilización de criterios en el WHERE son diferentes a los que se están usando en las funciones agregadas, el índice es muy costoso de utilizar y eventualmente no vale la pena crearlo porque la reducción del tiempo de consulta no representa un valor significativo o justifica el mantenimiento del índice.
   Tipo de índice utilizado: ninguno

6. **Justificación:** para el RFC6 no consideramos pertinente la creación de un índice porque para obtener la respuesta se deben revisar todos los registros de múltiples columnas y un índice no puede disminuir la cantidad de datos que se deben revisar. Además, las columnas involucradas en las consultas que responden a este RFC son actualizadas constantemente y el costo de mantenimiento del índice es injustificablemente alto.
   Tipo de índice utilizado: ninguno

7. **Justificación:** Para el RFC7 no consideramos pertinente la creación de un índice porque la consulta que cumple con este RFC hace uso de funciones agregadas para hacer la suma del dinero gastado por el cliente o el tiempo que el cliente se ha

hospedado en el hotel y simultáneamente necesita filtrar el año en el campo WHERE. Por este motivo, aunque un índice puede acelerar la consulta en algunos casos de uso de función agregada, la utilización de criterios en el WHERE diferentes a los que se están usando en las funciones agregadas hace que el índice sea muy costoso de utilizar y eventualmente no valga la pena crearlo porque la reducción del tiempo de consulta no representa un valor significativo o justifica el mantenimiento del índice.
Tipo de índice utilizado: ninguno

8. **Justificación:** en el RFC8 no consideramos pertinente la creación de un índice porque uno de los criterios más importantes de creación de índices es responder a la pregunta ¿Cuántas tuplas tengo que mirar para obtener la respuesta? En este caso particular, es necesario recorrer todos y cada uno de los registros independientemente de la creación de un índice, por lo que no se justifica asumir el costo de su creación.
   Tipo de índice utilizado: ninguno

9. **Justificación:** en el RFC9 creemos que es pertinente crear un índice en la tabla ReservasServicios, específicamente en el atributo fecha. Esta decisión se basa en i) pronosticamos que la selectividad de este atributo va a ser alta dado que hay muchas fechas en el calendario y es altamente probable que ¾ de los registros tengan fechas únicas, ii) evita el escaneo completo de la tabla y reduce el número de registros que se deben consultar, iii) asumimos que se va a necesitar hacer esta consulta múltiples veces, por lo que la eficiencia de búsqueda justifica la pérdida de eficiencia de creación del índice.
   Tipo de índice utilizado: árbol B+ por su eficiencia en la búsqueda de rangos

10. **Justificación:** en el RFC10 no consideramos pertinente la creación de un índice porque una de las condiciones que se va a aplicar es de la forma NOT IN y un índice no agilizaría significativamente la ardua comparación que implica este procedimiento.
    Tipo de índice utilizado: ninguno

11. **Justificación:** en el RFC11 no consideramos pertinente crear un índice porque la naturaleza de la pregunta hace necesario el recorrido de todos los registros en las columnas involucradas en las consultas. Esto se debe a que no es posible saber que, por ejemplo, un servicio es menos consumido que otro a menos que se compare con todos los existentes en la tabla. Adicionalmente, como las tablas son actualizadas constantemente el costo de mantener el índice es muy alto. Como uno de los criterios de creación de índices implícito más importante es el aumento de eficiencia, no es posible justificar el costo de creación y de mantenimiento de un índice que no reduce los registros sobre los que se hace la búsqueda.
    Tipo de índice utilizado: ninguno

12. **Justificación:** en el RFC12 se cree pertinente la creación de los siguientes índices:

- Índice simple sobre el atributo usuarios_idUser en la tabla RESERVAS. Este conjunto de datos tiene una selectividad óptima del 100% y acceder a ellos a través de un índice reduce el tiempo de búsqueda de datos
- Índice compuesto sobre el atributo checkIn y checkOut por la alta selectividad que presentan los valores de fechas, lo cual reduce el tiempo de la consulta al disminuir el tiempo de acceso a este valor.

Sin embargo, no se considera oportuno crear índices sobre el atributo costo ni duración de la tabla SERVICIOS. Esto se debe a que la selectividad de estos valores es muy baja pues en miles de datos no habrán duraciones o costos de un servicio que varíen en gran cantidad.

Tipo de índice utilizado: árbol B+, ideal para índices secundarios

ÍNDICES GENERADOS POR ORACLE

Información de los índices existentes tras la creación de las tablas:

OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS
ISIS23...	CONSUMOS_PK	NORMAL	ISIS2304C0220...	CONSUMOS	TABLE	UNIQUE
ISIS23...	EMPLEADOS_PK	NORMAL	ISIS2304C0220...	EMPLEADOS	TABLE	UNIQUE
ISIS23...	HABITACIONES_PK	NORMAL	ISIS2304C0220...	HABITACIONES	TABLE	UNIQUE
ISIS23...	PLANES_PK	NORMAL	ISIS2304C0220...	PLANES	TABLE	UNIQUE
ISIS23...	PLANESSERVICIOS_PK	NORMAL	ISIS2304C0220...	PLANESSERVICIOS	TABLE	UNIQUE
ISIS23...	RESERVAS_PK	NORMAL	ISIS2304C0220...	RESERVAS	TABLE	UNIQUE
ISIS23...	RESERVASSERVICIOS_PK	NORMAL	ISIS2304C0220...	RESERVASSERVICIOS	TABLE	UNIQUE
ISIS23...	SERVICIOS_PK	NORMAL	ISIS2304C0220...	SERVICIOS	TABLE	UNIQUE
ISIS23...	TIPOH_PK	NORMAL	ISIS2304C0220...	TIPOH	TABLE	UNIQUE
ISIS23...	TIPOSUSUARIOS_PK	NORMAL	ISIS2304C0220...	TIPOSUSUARIOS	TABLE	UNIQUE
ISIS23...	USUARIOS_PK	NORMAL	ISIS2304C0220...	USUARIOS	TABLE	UNIQUE

ÍNDICES A LLAVES PRIMARIAS:

RESERVAS:

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS
1	ISIS2304C25202320	RESERVAS_PK	NORMAL	ISIS2304C25202320	RESERVAS	TABLE	UNIQUE

Los índices cuyo nombre terminan en _PK son índices generados por Oracle sobre las llaves primarias de cada una de las tablas de la base de datos. Esta convención se genera porque las llaves primarias, dado que siempre son valores diferentes entre sí, tienen una selectividad del 100% y al ser los identificadores principales de una tupla un índice en la llave primaria agiliza la búsqueda de cualquier registro y mantiene la integridad de los datos al reforzar la característica UNIQUE de las llaves primarias.

Estos índices van a ayudar al desempeño de cualquier RFC que:

i) Incluya la unión de 2 tablas como parte del requerimiento, pues un índice organiza los datos y agiliza las comparaciones que se deben hacer para hacer un JOIN entre 2 tablas indexadas
ii) Filtre los datos bajo un criterio perteneciente a la llave primaria, pues puede llegar a reducir el número de registros que deben recorrerse. Esto ese especialmente cierto cuando el filtro que se va a aplicar es sobre un rango de valores
iii) Actualización de los registros, pues estos pueden ser ubicados con mayor rapidez

Esto se va a ver reflejado en el desempeño de RFC tales como el RFC1, el RFC3 y el RFC5 que realizan algunas de las operaciones mencionadas anteriormente.

CONSUMOS:

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS
1	ISIS2304C25202320	CONSUMOS_PK	NORMAL	ISIS2304C25202320	CONSUMOS	TABLE	UNIQUE

Los índices cuyo nombre terminan en _PK son índices generados por Oracle sobre las llaves primarias de cada una de las tablas de la base de datos. Esta convención se genera porque las llaves primarias, dado que siempre son valores diferentes entre sí, tienen una selectividad del 100% y al ser los identificadores principales de una tupla un índice en la llave primaria agiliza la búsqueda de cualquier registro y mantiene la integridad de los datos al reforzar la característica UNIQUE de las llaves primarias.

Estos índices van a ayudar al desempeño de cualquier RFC que:
i) Incluya la unión de 2 tablas como parte del requerimiento, pues un índice organiza los datos y agiliza las comparaciones que se deben hacer para hacer un JOIN entre 2 tablas indexadas
ii) Filtre los datos bajo un criterio perteneciente a la llave primaria, pues puede llegar a reducir el número de registros que deben recorrerse. Esto ese especialmente cierto cuando el filtro que se va a aplicar es sobre un rango de valores
iii) Actualización de los registros, pues estos pueden ser ubicados con mayor rapidez

Esto se va a ver reflejado en el desempeño de RFC tales como el RFC5 y el RFC7 que realizan algunas de las operaciones mencionadas anteriormente.

EMPLEADOS:

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS
1	ISIS2304C25202320	EMPLEADOS_PK	NORMAL	ISIS2304C25202320	EMPLEADOS	TABLE	UNIQUE

Los índices cuyo nombre terminan en _PK son índices generados por Oracle sobre las llaves primarias de cada una de las tablas de la base de datos. Esta convención se genera porque las llaves primarias, dado que siempre son valores diferentes entre sí, tienen una selectividad del 100% y al ser los identificadores principales de una tupla un índice en la

llave primaria agiliza la búsqueda de cualquier registro y mantiene la integridad de los datos al reforzar la característica UNIQUE de las llaves primarias.

Estos índices van a ayudar al desempeño de cualquier RFC que:

i) Incluya la unión de 2 tablas como parte del requerimiento, pues un índice organiza los datos y agiliza las comparaciones que se deben hacer para hacer un JOIN entre 2 tablas indexadas
ii) Filtre los datos bajo un criterio perteneciente a la llave primaria, pues puede llegar a reducir el número de registros que deben recorrerse. Esto ese especialmente cierto cuando el filtro que se va a aplicar es sobre un rango de valores
iii) Actualización de los registros, pues estos pueden ser ubicados con mayor rapidez

Esto no se va a ver reflejado necesariamente en los RFC que se tienen en el momento, pero si podría influir en próximos.

HABITACIONES:

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS
1	ISIS2304C25202320	HABITACIONES_PK	NORMAL	ISIS2304C25202320	HABITACIONES	TABLE	UNIQUE

Los índices cuyo nombre terminan en _PK son índices generados por Oracle sobre las llaves primarias de cada una de las tablas de la base de datos. Esta convención se genera porque las llaves primarias, dado que siempre son valores diferentes entre sí, tienen una selectividad del 100% y al ser los identificadores principales de una tupla un índice en la llave primaria agiliza la búsqueda de cualquier registro y mantiene la integridad de los datos al reforzar la característica UNIQUE de las llaves primarias.

Estos índices van a ayudar al desempeño de cualquier RFC que:

i) Incluya la unión de 2 tablas como parte del requerimiento, pues un índice organiza los datos y agiliza las comparaciones que se deben hacer para hacer un JOIN entre 2 tablas indexadas
ii) Filtre los datos bajo un criterio perteneciente a la llave primaria, pues puede llegar a reducir el número de registros que deben recorrerse. Esto ese especialmente cierto cuando el filtro que se va a aplicar es sobre un rango de valores
iii) Actualización de los registros, pues estos pueden ser ubicados con mayor rapidez

Esto se va a ver reflejado en el desempeño de RFC tales como el RFC1 y el RFC7 que realizan algunas de las operaciones mencionadas anteriormente.

PLANES:

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS
1	ISIS2304C25202320	PLANES_PK	NORMAL	ISIS2304C25202320	PLANES	TABLE	UNIQUE

Los índices cuyo nombre terminan en _PK son índices generados por Oracle sobre las llaves primarias de cada una de las tablas de la base de datos. Esta convención se genera porque las llaves primarias, dado que siempre son valores diferentes entre sí, tienen una selectividad del 100% y al ser los identificadores principales de una tupla un índice en la llave primaria agiliza la búsqueda de cualquier registro y mantiene la integridad de los datos al reforzar la característica UNIQUE de las llaves primarias.

Estos índices van a ayudar al desempeño de cualquier RFC que:

i) Incluya la unión de 2 tablas como parte del requerimiento, pues un índice organiza los datos y agiliza las comparaciones que se deben hacer para hacer un JOIN entre 2 tablas indexadas

ii) Filtre los datos bajo un criterio perteneciente a la llave primaria, pues puede llegar a reducir el número de registros que deben recorrerse. Esto ese especialmente cierto cuando el filtro que se va a aplicar es sobre un rango de valores

iii) Actualización de los registros, pues estos pueden ser ubicados con mayor rapidez

Esto no se va a ver reflejado necesariamente en los RFC que se tienen en el momento, pero si podría influir en próximos e influye en algunos RFC pasados.

PLANESSERVICIOS:

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENI
1	ISIS2304C25202320	PLANESSERVICIOS_PK	NORMAL	ISIS2304C25202320	PLANESSERVICIOS	TABLE	UNIQUE

Los índices cuyo nombre terminan en _PK son índices generados por Oracle sobre las llaves primarias de cada una de las tablas de la base de datos. Esta convención se genera porque las llaves primarias, dado que siempre son valores diferentes entre sí, tienen una selectividad del 100% y al ser los identificadores principales de una tupla un índice en la llave primaria agiliza la búsqueda de cualquier registro y mantiene la integridad de los datos al reforzar la característica UNIQUE de las llaves primarias.

Estos índices van a ayudar al desempeño de cualquier RFC que:

i) Incluya la unión de 2 tablas como parte del requerimiento, pues un índice organiza los datos y agiliza las comparaciones que se deben hacer para hacer un JOIN entre 2 tablas indexadas

ii) Filtre los datos bajo un criterio perteneciente a la llave primaria, pues puede llegar a reducir el número de registros que deben recorrerse. Esto ese especialmente cierto cuando el filtro que se va a aplicar es sobre un rango de valores

iii) Actualización de los registros, pues estos pueden ser ubicados con mayor rapidez

Esto no se va a ver reflejado necesariamente en los RFC que se tienen en el momento, pero si podría influir en próximos e influye en algunos RFC pasados.

RESERVASSERVICIOS:

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS
1	ISIS2304C25202320	RESERVASSERVICIOS_PK	NORMAL	ISIS2304C25202320	RESERVASSERVICIOS	TABLE	UNIQUE

Los índices cuyo nombre terminan en _PK son índices generados por Oracle sobre las llaves primarias de cada una de las tablas de la base de datos. Esta convención se genera porque las llaves primarias, dado que siempre son valores diferentes entre sí, tienen una selectividad del 100% y al ser los identificadores principales de una tupla un índice en la llave primaria agiliza la búsqueda de cualquier registro y mantiene la integridad de los datos al reforzar la característica UNIQUE de las llaves primarias.

Estos índices van a ayudar al desempeño de cualquier RFC que:

i) Incluya la unión de 2 tablas como parte del requerimiento, pues un índice organiza los datos y agiliza las comparaciones que se deben hacer para hacer un JOIN entre 2 tablas indexadas
ii) Filtre los datos bajo un criterio perteneciente a la llave primaria, pues puede llegar a reducir el número de registros que deben recorrerse. Esto ese especialmente cierto cuando el filtro que se va a aplicar es sobre un rango de valores
iii) Actualización de los registros, pues estos pueden ser ubicados con mayor rapidez

Esto se va a ver reflejado en el desempeño de RFC tales como el RFC4, el RFC 5, el RFC7, el RFC8, el RFC9, el RFC10 y el RFC12 que realizan algunas de las operaciones mencionadas anteriormente.

SERVICIOS:

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS	COMPRESSION
1	ISIS2304C25202320	SERVICIOS_PK	NORMAL	ISIS2304C25202320	SERVICIOS	TABLE	UNIQUE	DISABLED

Los índices cuyo nombre terminan en _PK son índices generados por Oracle sobre las llaves primarias de cada una de las tablas de la base de datos. Esta convención se genera porque las llaves primarias, dado que siempre son valores diferentes entre sí, tienen una selectividad del 100% y al ser los identificadores principales de una tupla un índice en la llave primaria agiliza la búsqueda de cualquier registro y mantiene la integridad de los datos al reforzar la característica UNIQUE de las llaves primarias.

Estos índices van a ayudar al desempeño de cualquier RFC que:

i) Incluya la unión de 2 tablas como parte del requerimiento, pues un índice organiza los datos y agiliza las comparaciones que se deben hacer para hacer un JOIN entre 2 tablas indexadas

ii) Filtre los datos bajo un criterio perteneciente a la llave primaria, pues puede llegar a reducir el número de registros que deben recorrerse. Esto ese especialmente cierto cuando el filtro que se va a aplicar es sobre un rango de valores
iii) Actualización de los registros, pues estos pueden ser ubicados con mayor rapidez

Esto se va a ver reflejado en el desempeño de RFC tales como el RFC12, el RFC 1, el RFC2 y el RFC4 que realizan algunas de las operaciones mencionadas anteriormente.

TIPOSH:

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS	COMPRESSION
1	ISIS2304C25202320	TIPOSH_PK	NORMAL	ISIS2304C25202320	TIPOSH	TABLE	UNIQUE	DISABLED

Los índices cuyo nombre terminan en _PK son índices generados por Oracle sobre las llaves primarias de cada una de las tablas de la base de datos. Esta convención se genera porque las llaves primarias, dado que siempre son valores diferentes entre sí, tienen una selectividad del 100% y al ser los identificadores principales de una tupla un índice en la llave primaria agiliza la búsqueda de cualquier registro y mantiene la integridad de los datos al reforzar la característica UNIQUE de las llaves primarias.

Estos índices van a ayudar al desempeño de cualquier RFC que:
i) Incluya la unión de 2 tablas como parte del requerimiento, pues un índice organiza los datos y agiliza las comparaciones que se deben hacer para hacer un JOIN entre 2 tablas indexadas
ii) Filtre los datos bajo un criterio perteneciente a la llave primaria, pues puede llegar a reducir el número de registros que deben recorrerse. Esto ese especialmente cierto cuando el filtro que se va a aplicar es sobre un rango de valores
iii) Actualización de los registros, pues estos pueden ser ubicados con mayor rapidez

Esto se va a ver reflejado en el desempeño de RFC tales como el RFC7 que realiza algunas de las operaciones mencionadas anteriormente.

TIPOSUSUARIOS:

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS	COMPRESSION
1	ISIS2304C25202320	TIPOSUSUARIOS_PK	NORMAL	ISIS2304C25202320	TIPOSUSUARIOS	TABLE	UNIQUE	DISABLED

Los índices cuyo nombre terminan en _PK son índices generados por Oracle sobre las llaves primarias de cada una de las tablas de la base de datos. Esta convención se genera porque las llaves primarias, dado que siempre son valores diferentes entre sí, tienen una

selectividad del 100% y al ser los identificadores principales de una tupla un índice en la llave primaria agiliza la búsqueda de cualquier registro y mantiene la integridad de los datos al reforzar la característica UNIQUE de las llaves primarias.

Estos índices van a ayudar al desempeño de cualquier RFC que:

i) Incluya la unión de 2 tablas como parte del requerimiento, pues un índice organiza los datos y agiliza las comparaciones que se deben hacer para hacer un JOIN entre 2 tablas indexadas
ii) Filtre los datos bajo un criterio perteneciente a la llave primaria, pues puede llegar a reducir el número de registros que deben recorrerse. Esto ese especialmente cierto cuando el filtro que se va a aplicar es sobre un rango de valores
iii) Actualización de los registros, pues estos pueden ser ubicados con mayor rapidez

Esto no se va a ver reflejado necesariamente en los RFC que se tienen en el momento, pero si podría influir en próximos e influye en algunos RFC pasados.

USUARIOS:

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS	COMPRESSION
1	ISIS2304C25202320	USUARIOS_PK	NORMAL	ISIS2304C25202320	USUARIOS	TABLE	UNIQUE	DISABLED

Los índices cuyo nombre terminan en _PK son índices generados por Oracle sobre las llaves primarias de cada una de las tablas de la base de datos. Esta convención se genera porque las llaves primarias, dado que siempre son valores diferentes entre sí, tienen una selectividad del 100% y al ser los identificadores principales de una tupla un índice en la llave primaria agiliza la búsqueda de cualquier registro y mantiene la integridad de los datos al reforzar la característica UNIQUE de las llaves primarias.

Estos índices van a ayudar al desempeño de cualquier RFC que:

iv) Incluya la unión de 2 tablas como parte del requerimiento, pues un índice organiza los datos y agiliza las comparaciones que se deben hacer para hacer un JOIN entre 2 tablas indexadas
v) Filtre los datos bajo un criterio perteneciente a la llave primaria, pues puede llegar a reducir el número de registros que deben recorrerse. Esto ese especialmente cierto cuando el filtro que se va a aplicar es sobre un rango de valores
vi) Actualización de los registros, pues estos pueden ser ubicados con mayor rapidez

Esto se va a ver reflejado en el desempeño de RFC tales como el RFC7, el RFC 9 y el RFC10 que realizan algunas de las operaciones mencionadas anteriormente.

DISEÑO DE CONSULTAS

Las consultas SQL están anexadas en un archivo .SQL dentro del repositorio

Análisis de distribución:

-RF1:

Rango de Fechas: Uno de los parámetros clave en esta consulta es el rango de fechas utilizado en la cláusula WHERE. El rango de fechas es r.fechaentrada >= TRUNC(SYSDATE, 'YYYY') - INTERVAL '1' YEAR, lo que significa que se están considerando las reservas realizadas durante el último año calendario. El tamaño de la respuesta variará significativamente según el rango de fechas elegido.

Número de Habitaciones y Servicios: El tamaño de la respuesta también dependerá del número de habitaciones y servicios disponibles en la base de datos.

-RF2:

Rango de Fechas: El rango de fechas definido en la cláusula WHERE es crucial. Los valores '2023-01-01' y '2023-11-02' representan el inicio y el final del período analizado. El tamaño de la respuesta variará según el rango de fechas elegido. Si se acorta o se amplía el rango, afectará directamente la cantidad de registros incluidos en el resultado. Un rango más amplio incluirá más consumos, mientras que un rango más corto limitará la cantidad de consumos en la respuesta.

Numero de Servicios y ReservasServicios: La cantidad de registros en las tablas servicios y reservasservicios en los datos de prueba influirá en el tamaño de la respuesta. Si hay una gran cantidad de registros, la respuesta será más grande en comparación con una base de datos con menos registros.

-RF3:

Rango de Fechas: El rango de fechas definido en la cláusula WHERE es fundamental. El rango es r.fechasalida >= (SYSDATE - 365), lo que significa que se están considerando las reservas cuya fecha de salida está dentro del último año (365 días) a partir de la fecha actual (SYSDATE). El tamaño de la respuesta variará según el rango de fechas elegido

Habitaciones y reservas: La cantidad de registros en las tablas habitaciones y reservas en los datos de prueba influirá en el tamaño de la respuesta. Si hay una gran cantidad de registros de reservas, la respuesta será más grande en comparación con una base de datos con menos registros de reservas.

-RF4:

Rango de Fechas: El rango de fechas definido en la cláusula WHERE es fundamental. El rango es rs.fecha BETWEEN TO_DATE('2023-01-01', 'YYYY-MM-DD') AND TO_DATE('2023-11-05', 'YYYY-MM-DD'). Esto significa que se están considerando los registros de reservasservicios cuyas fechas están dentro de este período. El tamaño de la respuesta variará según el rango de fechas elegido. Cambiar las fechas de inicio y final afectará el número de registros incluidos en el resultado.

Costo del Servicio: La condición s.costo BETWEEN 0 AND 16000000 filtra los servicios con un costo dentro de este rango. Modificar el rango de costos afectará la cantidad de servicios que cumplan con este criterio. Si se amplía el rango, se incluirán servicios más costosos en la respuesta.

Nombre del Servicio: La condición s.nombreservicio = 'Gimnasio' filtra los registros que coinciden con el nombre de servicio 'Gimnasio'. Cambiar el nombre del servicio afectará los resultados, ya que solo se incluirán registros relacionados con el servicio especificado.

-RF5:

Nombre del Usuario: El valor 'Esteban' en la condición u.nombreusuario = 'Esteban' filtra las reservas y consumos asociados al usuario con este nombre. Cambiar el nombre del usuario afectará los resultados, ya que se incluirán registros relacionados con el usuario especificado.

Rango de Fechas: El rango de fechas definido en las condiciones c.fecha >= TO_DATE('2023-01-01', 'YYYY-MM-DD') y c.fecha <= TO_DATE('2023-11-05', 'YYYY-MM-DD') afecta los consumos incluidos en la respuesta. Cambiar las fechas de inicio y final afectará el período de tiempo que se considera para los consumos. Si se amplía o reduce el rango de fechas, afectará la cantidad de consumos en la respuesta.

-RF6

Fecha entrada: La cantidad de valores de fecha entrada de la tabla reservas afecta los resultados. Entre más reservas haya más registros habrá

Consumos: La cantidad de consumos realizados afecta directamente en los resultados de respuesta, entre más registros de consumo, más valores de fechas estarán en la tabla, afectando así los resultados.

-RF7

El criterio de "Ha estado en el hotel por al menos dos semanas" se basa en la diferencia entre las fechas de entrada y salida en las reservas. Los parámetros clave en este caso son las fechas

utilizadas en la condición r.fechasalida >= TO_DATE('2022-11-05', 'YYYY-MM-DD') - INTERVAL '1' YEAR para calcular la estadía de al menos dos semanas. Cambiar la fecha de referencia (por ejemplo, cambiar el '1 YEAR' a '6 MONTHS') afectará los resultados y el número de "buenos clientes" identificados

El criterio de "Su consumo total es mayor a 15.000.000" se basa en la suma de los costos de consumo de los servicios y el costo de las habitaciones. Los parámetros clave en este caso son el umbral de 15.000.000 y las tablas involucradas en el JOIN. Cambiar el umbral de consumo afectará el número de "buenos clientes" identificados. Además, la cantidad de datos en las tablas usuarios, reservas, habitaciones, tiposh, reservasservicios, servicios y consumos influirá en los resultados.

-RF8

El criterio es "Encontrar los servicios que hayan sido solicitados menos de 3 veces semanales, durante el último año de operación de HotelAndes." Los parámetros clave en este caso son el número "3" (umbral de demanda) y la fecha utilizada para el último año de operación (SYSDATE - 365). Cambiar el umbral de demanda o ajustar la fecha afectará los resultados y el número de servicios identificados con poca demanda.

-RF9

Criterio de Consumo de Servicio : El criterio es "clientes que consumieron al menos una vez un determinado servicio del hotel" en un rango de fechas. Los parámetros clave en este caso son:
*El nombre del servicio
*El rango de fechas utilizado para filtrar las reservas.

Cambiar el nombre del servicio o el rango de fechas afectará los resultados y el número de clientes identificados.

-RF10

El criterio es "clientes que no consumieron el servicio de Piscina" en un rango de fechas. Los parámetros clave en este caso son:
*El nombre del servicio
*El rango de fechas utilizado para filtrar las reservas.

Cambiar el nombre del servicio o el rango de fechas afectará los resultados y el número de clientes identificados.

-RF12

La consulta tiene dos secciones con diferentes criterios para identificar a un "Excelente Cliente". Los criterios clave en este caso son:
*Para la primera sección: alojarse al menos una vez por trimestre.
*Para la segunda sección: tener reservas de SPA, salones de reuniones con duración > 4 horas o reservas con costo total >= 300,000.
Cambiar estos criterios afectará los resultados y el número de clientes identificados como "Excelentes Clientes".

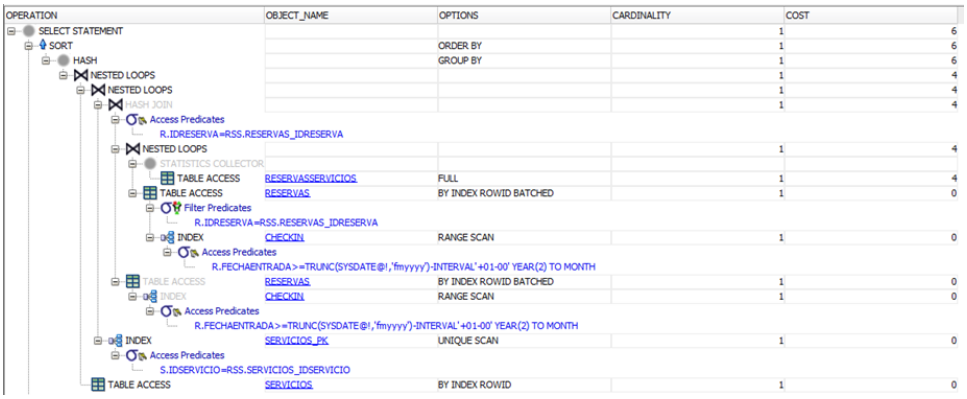
La cantidad de registros en las tablas usuarios, reservas, reservasservicios, y servicios influirá en los resultados, entre más registros haya, más afectará a los resultados y al tiempo de ejecución de la consulta.

Valores de parámetros utilizados en el análisis:

```
INSERT INTO tiposh VALUES ('Familiar', 'Tiene 1 minifridge, 2 camas king, 2 banos, 1 televisor', 5, 50000);
INSERT INTO tiposh VALUES ('SuitePresidencial', 'Tiene 3 minifridge, 4 camas king, 4 banos, 3 televisor', 7, 80000);
INSERT INTO habitaciones VALUES (701, 'Familiar');
INSERT INTO habitaciones VALUES (702, 'SuitePresidencial');
INSERT INTO habitaciones VALUES (703, 'Familiar');
INSERT INTO habitaciones VALUES (704, 'SuitePresidencial');
INSERT INTO habitaciones VALUES (705, 'Familiar');
INSERT INTO habitaciones VALUES (706, 'SuitePresidencial');
INSERT INTO habitaciones VALUES (707, 'SuitePresidencial');
INSERT INTO habitaciones VALUES (708, 'SuitePresidencial');
INSERT INTO habitaciones VALUES (709, 'Familiar');
INSERT INTO habitaciones VALUES (710, 'Familiar');
INSERT INTO habitaciones VALUES (711, 'Familiar');
INSERT INTO habitaciones VALUES (712, 'SuitePresidencial');
INSERT INTO servicios VALUES (1, 'Piscina', 'Piscina', '6:00 AM - 11:00 PM', 100, 5000);
INSERT INTO servicios VALUES (2, 'Gimnasio', 'Gimnasio', '6:00 AM - 11:00 PM', 300, 3000);
INSERT INTO servicios VALUES (3, 'Prueba15Millones', 'Prueba15Millones', '6:00 AM - 11:00 PM', 300, 16000000);
INSERT INTO servicios VALUES (4, 'SPA', 'SPA', '6:00 AM - 11:00 PM', 10, 400000);
INSERT INTO servicios VALUES (5, 'Salones de Reuniones', 'Salones de Reuniones', '6:00 AM - 11:00 PM', 120, 300000);
INSERT INTO empleados VALUES (100, 'Empleado');
INSERT INTO empleados VALUES (101, 'Empleado');
INSERT INTO empleados VALUES (102, 'Empleado');
INSERT INTO empleados VALUES (104, 'Empleado');
INSERT INTO usuarios VALUES ('0000000005', 'CC', 'Esteban', 'esteban@uniandes.edu.co', 'Cliente');
INSERT INTO usuarios VALUES ('0000000006', 'CC', 'Juan', 'Juan@uniandes.edu.co', 'Cliente');
INSERT INTO usuarios VALUES ('0000000007', 'CC', 'Jose', 'Jose@uniandes.edu.co', 'Cliente');
INSERT INTO usuarios VALUES ('0000000008', 'CC', 'Alejandro', 'Alejandro@uniandes.edu.co', 'Cliente');
INSERT INTO usuarios VALUES ('0000000009', 'CC', 'Andrea', 'Andrea@uniandes.edu.co', 'Cliente');
INSERT INTO usuarios VALUES ('0000000010', 'CC', 'Daniela', 'Daniela@uniandes.edu.co', 'Cliente');
INSERT INTO usuarios VALUES ('0000000011', 'CC', 'Cristian', 'Cristian@uniandes.edu.co', 'Cliente');
INSERT INTO reservas VALUES (1, TO_DATE('2023-01-01', 'YYYY-MM-DD'), TO_DATE('2023-01-10', 'YYYY-MM-DD'), 2, 'Basico', 0000000005, 701);
INSERT INTO reservas VALUES (2, TO_DATE('2023-01-01', 'YYYY-MM-DD'), TO_DATE('2023-01-10', 'YYYY-MM-DD'), 3, 'VIP', 0000000006, 702);
INSERT INTO reservas VALUES (3, TO_DATE('2023-01-11', 'YYYY-MM-DD'), TO_DATE('2023-01-15', 'YYYY-MM-DD'), 4, 'VIP', 0000000007, 703);
INSERT INTO reservas VALUES (4, TO_DATE('2023-01-12', 'YYYY-MM-DD'), TO_DATE('2023-02-12', 'YYYY-MM-DD'), 5, 'VIP', 0000000009, 704);
INSERT INTO reservas VALUES (5, TO_DATE('2023-02-11', 'YYYY-MM-DD'), TO_DATE('2023-03-11', 'YYYY-MM-DD'), 4, 'VIP', 0000000008, 705);
INSERT INTO reservas VALUES (6, TO_DATE('2023-02-15', 'YYYY-MM-DD'), TO_DATE('2023-02-16', 'YYYY-MM-DD'), 4, 'VIP', 0000000010, 706);
INSERT INTO reservas VALUES (7, TO_DATE('2023-05-15', 'YYYY-MM-DD'), TO_DATE('2023-05-16', 'YYYY-MM-DD'), 4, 'VIP', 0000000010, 707);
INSERT INTO reservas VALUES (8, TO_DATE('2023-03-11', 'YYYY-MM-DD'), TO_DATE('2023-03-15', 'YYYY-MM-DD'), 4, 'VIP', 0000000011, 708);
INSERT INTO reservas VALUES (9, TO_DATE('2023-04-01', 'YYYY-MM-DD'), TO_DATE('2023-04-10', 'YYYY-MM-DD'), 2, 'Basico', 0000000005, 709);
INSERT INTO reservas VALUES (10, TO_DATE('2023-07-01', 'YYYY-MM-DD'), TO_DATE('2023-07-10', 'YYYY-MM-DD'), 2, 'Basico', 0000000005, 710);
INSERT INTO reservas VALUES (11, TO_DATE('2023-10-01', 'YYYY-MM-DD'), TO_DATE('2023-10-10', 'YYYY-MM-DD'), 2, 'Basico', 0000000005, 711);
INSERT INTO reservas VALUES (12, TO_DATE('2023-10-01', 'YYYY-MM-DD'), TO_DATE('2023-10-10', 'YYYY-MM-DD'), 2, 'Basico', 0000000011, 712);
INSERT INTO reservas VALUES (13, TO_DATE('2023-01-20', 'YYYY-MM-DD'), TO_DATE('2023-01-22', 'YYYY-MM-DD'), 2, 'Basico', 0000000005, 701);
INSERT INTO consumos VALUES (1, 'Almuerzo piscina', TO_DATE('2023-10-30', 'YYYY-MM-DD'), 40, 1);
INSERT INTO consumos VALUES (2, 'Entrenamiento personalizado GYM', TO_DATE('2023-10-30', 'YYYY-MM-DD'), 20, 2);
INSERT INTO reservasservicios VALUES (1,TO_DATE('2023-01-04', 'YYYY-MM-DD'),5 , 1, 1, 101);
INSERT INTO reservasservicios VALUES (2, TO_DATE('2023-01-05', 'YYYY-MM-DD'),5, 2, 2, 101);
INSERT INTO reservasservicios VALUES (3, TO_DATE('2023-01-06', 'YYYY-MM-DD'),5, 2, 3, 102);
INSERT INTO reservasservicios VALUES (4, TO_DATE('2023-01-07', 'YYYY-MM-DD'),5, 3, 4, 104);
INSERT INTO reservasservicios VALUES (5, TO_DATE('2023-01-08', 'YYYY-MM-DD'),5, 2, 5, 104);
INSERT INTO reservasservicios VALUES (6, TO_DATE('2023-01-08', 'YYYY-MM-DD'),6, 4, 12, 104);
```

Planes de consulta obtenidos por Oracle:

RF1:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	6
SORT		ORDER BY	1	6
HASH		GROUP BY	1	6
NESTED LOOPS			1	4
NESTED LOOPS			1	4
HASH JOIN			1	4
Access Predicates				
R.IDRESERVA=RSS.RESERVAS_IDRESERVA				
NESTED LOOPS			1	4
STATISTICS COLLECTOR				
TABLE ACCESS	RESERVASSERVICIOS	FULL	1	4
TABLE ACCESS	RESERVAS	BY INDEX ROWID BATCHED	1	0
Filter Predicates				
R.IDRESERVA=RSS.RESERVAS_IDRESERVA				
INDEX	CHECKIN	RANGE SCAN	1	0
Access Predicates				
R.FECHAENTRADA>=TRUNC(SYSDATE@!,'fmyyyy')-INTERVAL'+01-00' YEAR(2) TO MONTH				
TABLE ACCESS	RESERVAS	BY INDEX ROWID BATCHED	1	0
INDEX	CHECKIN	RANGE SCAN	1	0
Access Predicates				
R.FECHAENTRADA>=TRUNC(SYSDATE@!,'fmyyyy')-INTERVAL'+01-00' YEAR(2) TO MONTH				
INDEX	SERVICIOS_PK	UNIQUE SCAN	1	0
Access Predicates				
S.IDSERVICIO=RSS.SERVICIOS_IDSERVICIO				
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	1	0

Se utiliza un ORDER BY con el fin de ordenar los resultados por número de habitación y nombre del servicio

Se utiliza un GROUP BY con el fin de ordenar los resultados por número de habitación y nombre del servicio

Se utiliza un FULL para acceder a la tabla reservaServicios, es decir, un recorrido completo de la tabla.

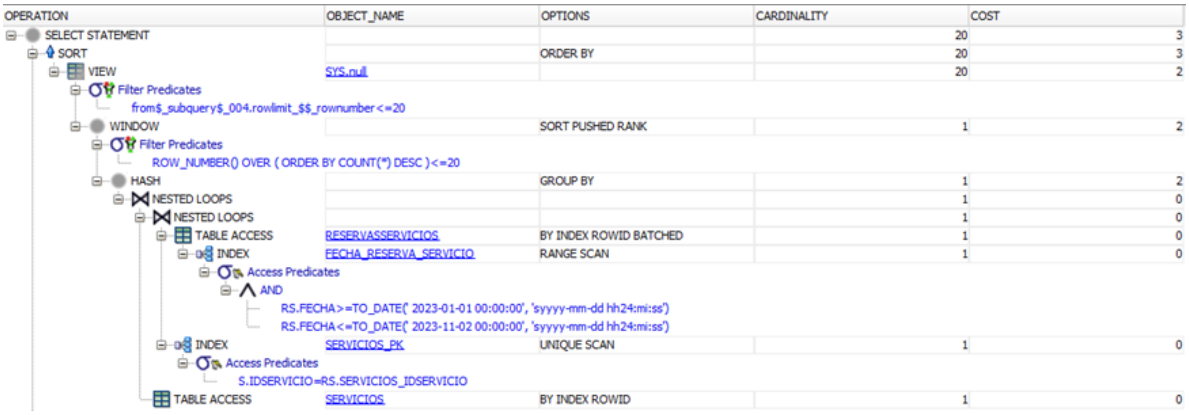
Se utiliza un By INDEX ROWID BATCHED para acceder a la tabla reservas 2 veces, lo que significa que está realizando una operación que implica una búsqueda y recuperación de datos por medio de índices

Se utiliza un RANGE SCAN 2 veces, que es una operación de acceso a datos que utiliza un índice para recuperar dentro un rango especifico de valores en la tabla, en este caso el valor CHECKIN

Se utiliza UNIQUE SCAN para recuperar una fila especifica en la tabla Servicios, en este caso, el ID

Se utiliza un BY INDEX ROWID para acceder a la tabla servicios con el fin de encontrar la coincidencia en s.idServicio y rss.servicios_idservicio.

RF2:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			20	3
SORT		ORDER BY	20	3
VIEW	SYS.null		20	2
Filter Predicates				
from$_subquery$_004.rowlimit_$$_rownumber<=20				
WINDOW		SORT PUSHED RANK	1	2
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC)<=20				
HASH		GROUP BY	1	2
NESTED LOOPS			1	0
NESTED LOOPS			1	0
TABLE ACCESS	RESERVASSERVICIOS	BY INDEX ROWID BATCHED	1	0
INDEX	FECHA_RESERVA_SERVICIO	RANGE SCAN	1	0
Access Predicates				
AND				
RS.FECHA>=TO_DATE(' 2023-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				
RS.FECHA<=TO_DATE(' 2023-11-02 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				
INDEX	SERVICIOS_PK	UNIQUE SCAN	1	0
Access Predicates				
S.IDSERVICIO=RS.SERVICIOS_IDSERVICIO				
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	1	0

Se utiliza un ORDER BY con el fin de ordenar los resultados por el total de consumos

Se utiliza un GROUP BY con el fin de agrupar los resultados por s.nombreservicio

Se utiliza 2 BY INDEX ROWID BATCHED para acceder a la tabla reservaservicios y servicios, lo que significa que está realizando una operación que implica una búsqueda y recuperación de datos por medio de índices

Se utiliza un RANGE SCAN que es una operación de acceso a datos que utiliza un índice para recuperar dentro un rango especifico de valores en la tabla, en este caso fecha_reserva_servicio

Se utiliza un UNIQUE SCAN para recuperar una fila especifica en la tabla servicio

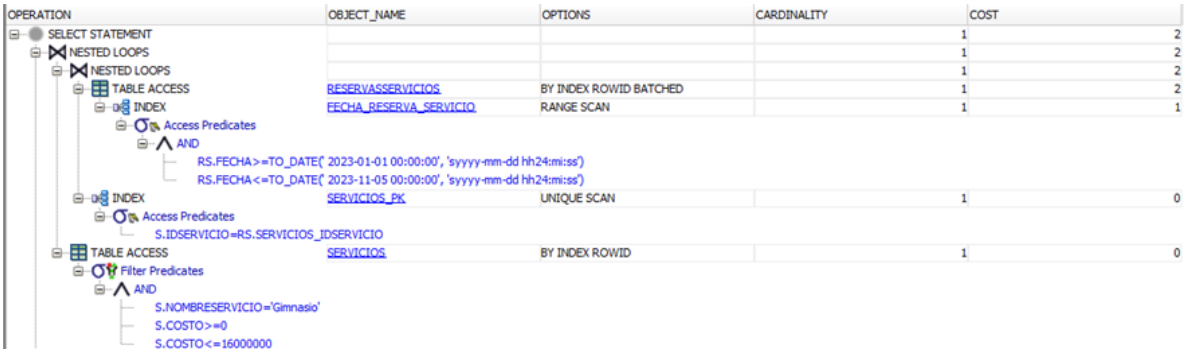
RF3:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	1
SORT		GROUP BY	1	1
TABLE ACCESS	RESERVAS	BY INDEX ROWID BATCHED	1	0
INDEX	CHECKOUT	RANGE SCAN	1	0
Access Predicates				
R.FECHASALIDA>=SYSDATE@!-365				

Se utiliza un GROUP BY con el fin de agrupar los resultados de la consulta por el numero de habitación o h.numero

Se utiliza un BY INDEX ROWID BATCHED para acceder a la tabla reservas, lo que significa que está realizando una operación que implica una búsqueda y recuperación de datos por medio de índices

RF4:



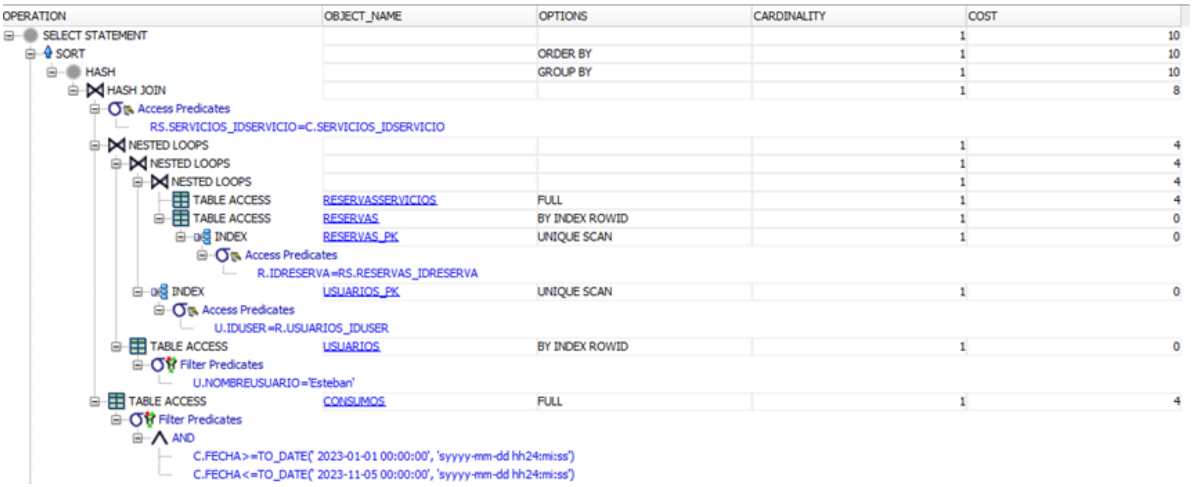
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	2
NESTED LOOPS			1	2
NESTED LOOPS			1	2
TABLE ACCESS	RESERVASSERVICIOS	BY INDEX ROWID BATCHED	1	2
INDEX	FECHA_RESERVA_SERVICIO	RANGE SCAN	1	1
Access Predicates				
AND				
RS.FECHA>=TO_DATE(' 2023-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				
RS.FECHA<=TO_DATE(' 2023-11-05 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				
INDEX	SERVICIOS_PK	UNIQUE SCAN	1	0
Access Predicates				
S.IDSERVICIO=RS.SERVICIOS_IDSERVICIO				
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	1	0
Filter Predicates				
AND				
S.NOMBRESERVICIO='Gimnasio'				
S.COSTO>=0				
S.COSTO<=16000000				

Se utiliza 2 BY INDEX ROWID BATCHED para acceder a la tabla reservasservicios y servicios, lo que significa que está realizando una operación que implica una búsqueda y recuperación de datos por medio de índices

Se utiliza un RANGE SCAN que es una operación de acceso a datos que utiliza un índice para recuperar dentro un rango especifico de valores en la tabla, en este caso fecha_reserva_servicio

Se utiliza un UNIQUE SCAN para recuperar una fila especifica en la tabla servicio

RF5:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	10
SORT		ORDER BY	1	10
HASH		GROUP BY	1	10
HASH JOIN			1	8
Access Predicates				
RS.SERVICIOS_IDSERVICIO=C.SERVICIOS_IDSERVICIO				
NESTED LOOPS			1	4
NESTED LOOPS			1	4
NESTED LOOPS			1	4
TABLE ACCESS	RESERVASSERVICIOS	FULL	1	4
TABLE ACCESS	RESERVAS	BY INDEX ROWID	1	0
INDEX	RESERVAS_PK	UNIQUE SCAN	1	0
Access Predicates				
R.IDRESERVA=RS.RESERVAS_IDRESERVA				
INDEX	USUARIOS_PK	UNIQUE SCAN	1	0
Access Predicates				
U.IDUSER=R.USUARIOS_IDUSER				
TABLE ACCESS	USUARIOS	BY INDEX ROWID	1	0
Filter Predicates				
U.NOMBREUSUARIO='Esteban'				
TABLE ACCESS	CONSUMOS	FULL	1	4
Filter Predicates				
AND				
C.FECHA>=TO_DATE(' 2023-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				
C.FECHA<=TO_DATE(' 2023-11-05 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				

Se utiliza un ORDER y GROUP by con el fin de ordenar y agrupar los resultados en base a id del usuario, el nombre y en TOTAL GASTADO EN COSUMOS

Se utiliza 2 FULL para acceder a la tabla reservaServicios y consumos, es decir, un recorrido completo de la tabla.

Se utiliza un BY INDEX ROWID para acceder a la tabla reservas, lo que significa que está realizando una operación que implica una búsqueda y recuperación de datos por medio de índices

Se utiliza 2 UNIQUE SCAN para recuperar una fila especifica en la tabla usuarios

RF6:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	9
SORT		ORDER BY	1	9
MERGE JOIN		CARTESIAN	1	8
MERGE JOIN		CARTESIAN	1	2
VIEW	SYS.null		1	1
Filter Predicates				
from$_subquery$_002.rowlimit_$$_rownumber<=1				
WINDOW		SORT PUSHED RANK	1	1
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC)<=1				
SORT		GROUP BY NOSORT	1	1
INDEX	CHECKIN	FULL SCAN	1	0
BUFFER		SORT	1	2
VIEW	SYS.null		1	1
Filter Predicates				
from$_subquery$_006.rowlimit_$$_rownumber<=1				
WINDOW		SORT PUSHED RANK	1	1
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*))<=1				
SORT		GROUP BY NOSORT	1	1
INDEX	CHECKIN	FULL SCAN	1	0
BUFFER		SORT	1	8
VIEW	SYS.null		1	6
Filter Predicates				
from$_subquery$_004.rowlimit_$$_rownumber<=1				
WINDOW		SORT PUSHED RANK	1	6
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY SUM(COSTO) DESC)<=1				
HASH		GROUP BY	1	6
TABLE ACCESS	CONSUMOS	FULL	1	4

Se utiliza un ORDER BY y ORDER BY con el fin de ordenar los resultados, esto depende de la subconsulta, en este caso para “Ocupación” se utiliza fechaentrada de reserva y cantidad_ocupaciones. Para “Ingresos” se utiliza la fecha de consumo e ingresos totales. Finalmente, para “Demanda” se utiliza fechaentrada de reserva y cantidad_ocupaciones

Se utiliza 2 FULL para acceder a la tabla consumos

RF7:

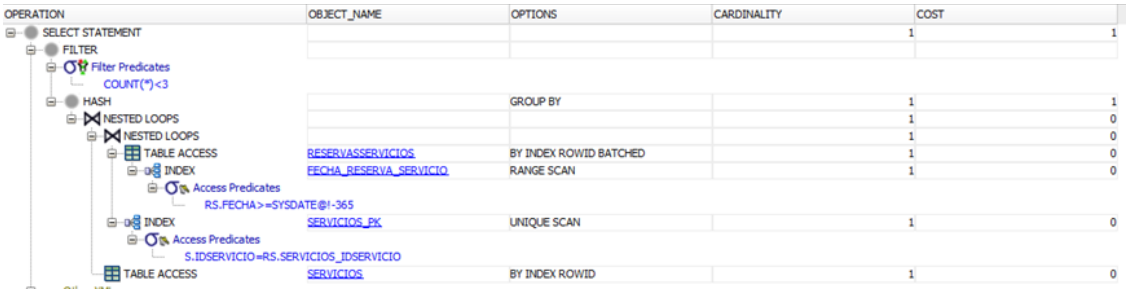
Se utiliza FULL para hacer un recorrido completo de USUARIOS en 2 ocasiones

Se utiliza GROUP BY para ordenar los resultados de la primera subconsulta por el id del usuario

Se utiliza 3 BY INDEX ROWID para acceder a la tabla reservas, habitaciones y servicios, lo que significa que está realizando una operación que implica una búsqueda y recuperación de datos por medio de índices

Se utiliza 2 UNIQUE SCAN para acceder a TiposH y Servicios que es un acceso a datos que utiliza un índice único para buscar y recuperar la fila

RF8:



Se utiliza un GROUP BY con el fin de agrupar los resultados en base al ID del servicio y el nombre del servicio

Se utiliza 2 BY INDEX ROWID para acceder a la tabla reservasservicios y servicios, lo que significa que está realizando una operación que implica una búsqueda y recuperación de datos por medio de índices

Se utiliza un UNIQUE SCAN para recuperar una fila especifica en la tabla servicio

RF9:

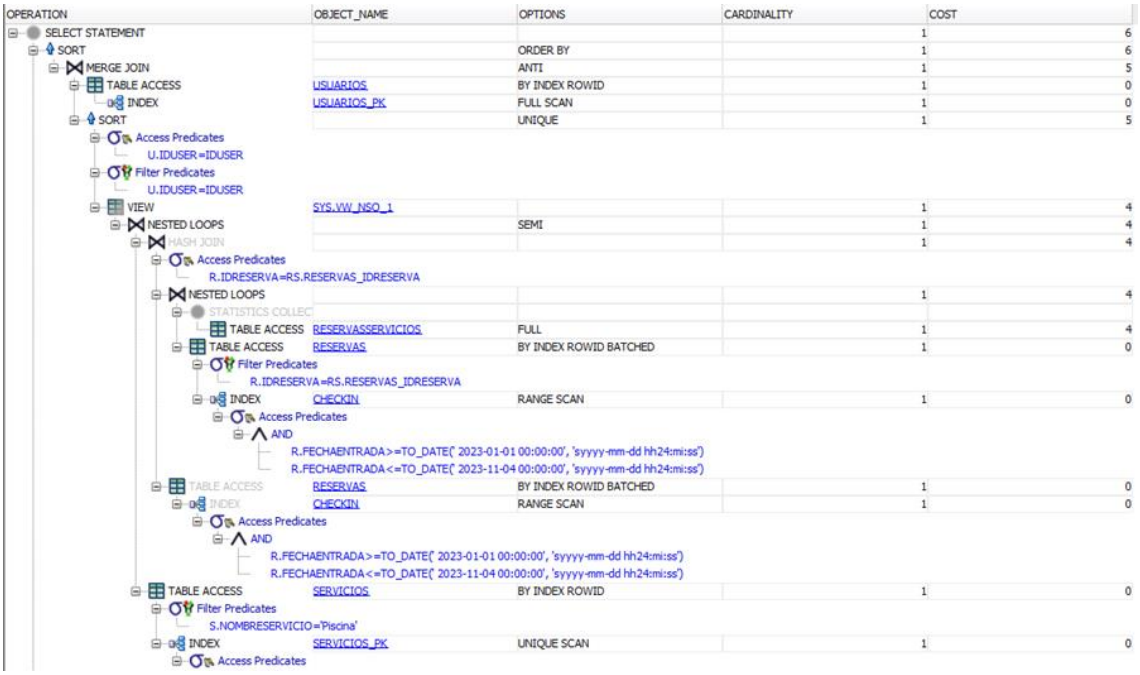
Se utiliza un ORDER BY y GROUP BY para order y agrupar los resultados de la consulta en base a el id del usuario, el tipo de documento, el correo y el nombre

Se utiliza un FULL para acceder a la tabla reservaServicios y consumos, es decir, un recorrido completo de la tabla.

Se utiliza un UNIQUE SCAN para recuperar una fila especifica en la tabla servicio.

Se utiliza 2 BY INDEX ROWID para acceder a la tabla servicios y usuarios, lo que significa que está realizando una operación que implica una búsqueda y recuperación de datos por medio de índices.

RF10:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	6
SORT		ORDER BY	1	6
MERGE JOIN		ANTI	1	5
TABLE ACCESS	USUARIOS	BY INDEX ROWID	1	0
INDEX	USUARIOS_PK	FULL SCAN	1	0
SORT		UNIQUE	1	5
Access Predicates				
U.IDUSER=IDUSER				
Filter Predicates				
U.IDUSER=IDUSER				
VIEW	SYS.VW_NSO_1		1	4
NESTED LOOPS		SEMI	1	4
HASH JOIN			1	4
Access Predicates				
R.IDRESERVA=RS.RESERVAS_IDRESERVA				
NESTED LOOPS			1	4
STATISTICS COLLEC				
TABLE ACCESS	RESERVASSERVICIOS	FULL	1	4
TABLE ACCESS	RESERVAS	BY INDEX ROWID BATCHED	1	0
Filter Predicates				
R.IDRESERVA=RS.RESERVAS_IDRESERVA				
INDEX	CHECKIN	RANGE SCAN	1	0
Access Predicates				
AND				
R.FECHAENTRADA>=TO_DATE(' 2023-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				
R.FECHAENTRADA<=TO_DATE(' 2023-11-04 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				
TABLE ACCESS	RESERVAS	BY INDEX ROWID BATCHED	1	0
INDEX	CHECKIN	RANGE SCAN	1	0
Access Predicates				
AND				
R.FECHAENTRADA>=TO_DATE(' 2023-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				
R.FECHAENTRADA<=TO_DATE(' 2023-11-04 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	1	0
Filter Predicates				
S.NOMBRESERVICIO='Piscina'				
INDEX	SERVICIOS_PK	UNIQUE SCAN	1	0
Access Predicates				

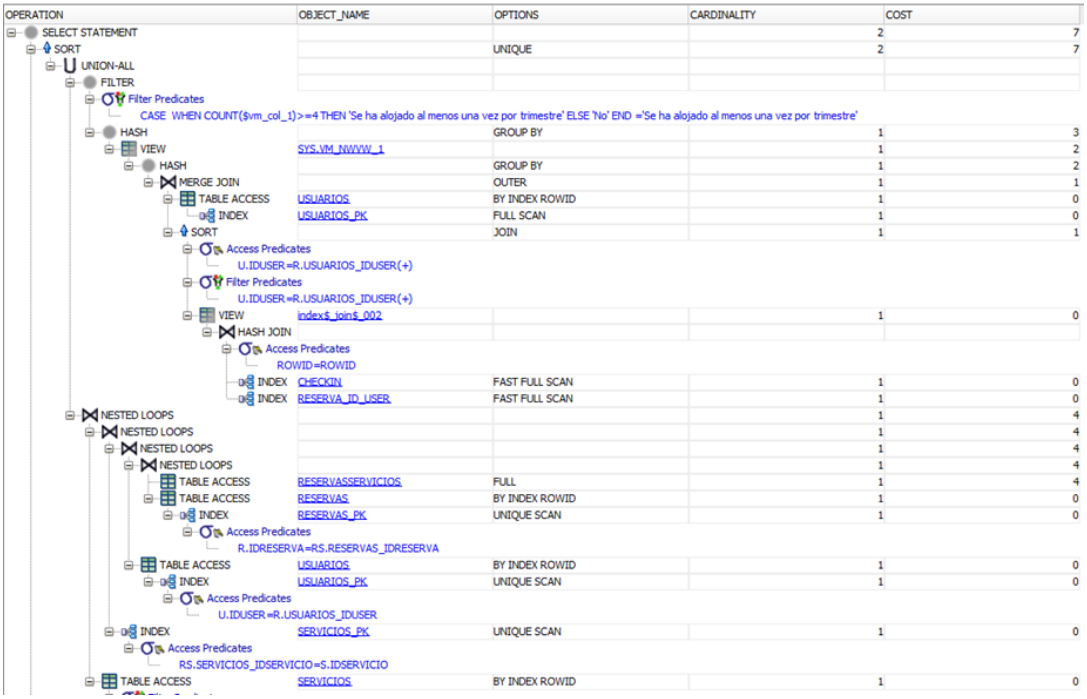
Se utiliza un ORDER BY con el fin de ordenar los resultados en base al nombre del usuario

Se utiliza 2 BY INDEX ROWID BATCHED para acceder a la tabla usuarios y reservas, lo que significa que está realizando una operación que implica una búsqueda y recuperación de datos por medio de índices

Se utiliza un FULL para acceder a la tabla reservaServicios, es decir, un recorrido completo de la tabla.

Se utiliza un UNIQUE SCAN para recuperar una fila especifica en la tabla servicio

RF12:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			2	7
SORT		UNIQUE	2	7
UNION-ALL				
FILTER				
Filter Predicates				
CASE WHEN COUNT($vm_col_1)>=4 THEN 'Se ha alojado al menos una vez por trimestre' ELSE 'No' END ='Se ha alojado al menos una vez por trimestre'				
HASH		GROUP BY	1	3
VIEW	SYS.VM_NWVW_1		1	2
HASH		GROUP BY	1	2
MERGE JOIN		OUTER	1	1
TABLE ACCESS	USUARIOS	BY INDEX ROWID	1	0
INDEX	USUARIOS_PK	FULL SCAN	1	0
SORT		JOIN	1	1
Access Predicates				
U.IDUSER=R.USUARIOS_IDUSER(+)				
Filter Predicates				
U.IDUSER=R.USUARIOS_IDUSER(+)				
VIEW	index$_join$_002		1	0
HASH JOIN				
Access Predicates				
ROWID=ROWID				
INDEX	CHECKIN	FAST FULL SCAN	1	0
INDEX	RESERVA_ID_USER	FAST FULL SCAN	1	0
NESTED LOOPS			1	4
NESTED LOOPS			1	4
NESTED LOOPS			1	4
NESTED LOOPS			1	4
TABLE ACCESS	RESERVASSERVICIOS	FULL	1	4
TABLE ACCESS	RESERVAS	BY INDEX ROWID	1	0
INDEX	RESERVAS_PK	UNIQUE SCAN	1	0
Access Predicates				
R.IDRESERVA=RS.RESERVAS_IDRESERVA				
TABLE ACCESS	USUARIOS	BY INDEX ROWID	1	0
INDEX	USUARIOS_PK	UNIQUE SCAN	1	0
Access Predicates				
U.IDUSER=R.USUARIOS_IDUSER				
INDEX	SERVICIOS_PK	UNIQUE SCAN	1	0
Access Predicates				
RS.SERVICIOS_IDSERVICIO=S.IDSERVICIO				
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	1	0

Se utiliza un GROUP BY con el fin de agrupar los resultados en base al id del usuario, el nombre del usuario, el tipo de documento y el correo

Se utiliza 3 BY INDEX ROWID para acceder a la tabla usuarios, reservas y servicios, lo que significa que está realizando una operación que implica una búsqueda y recuperación de datos por medio de índices

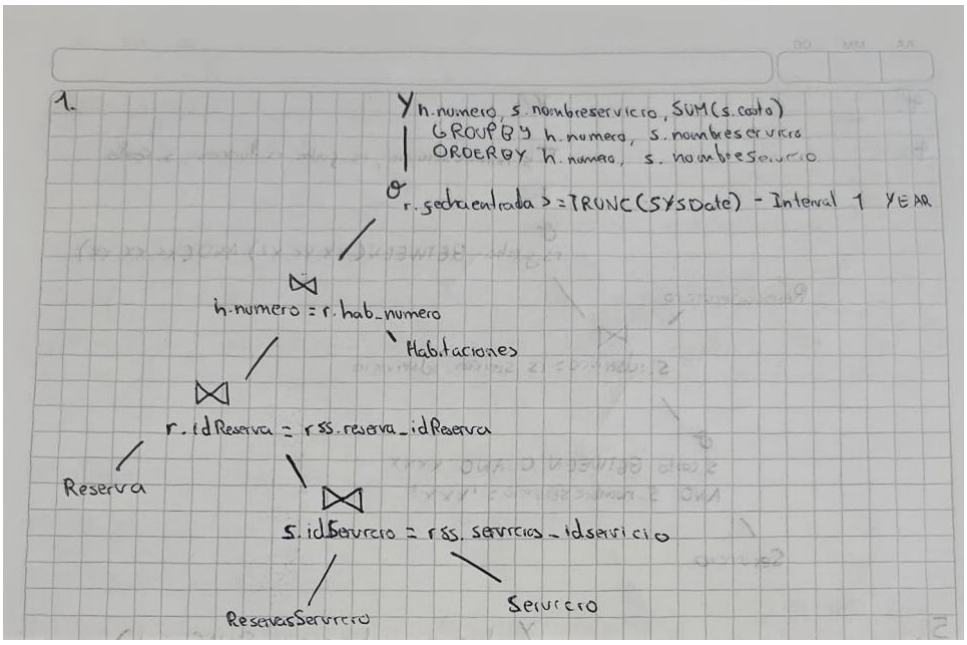
Se utiliza 2 FULL para acceder a la tabla reservaServicios y usuarios, es decir, un recorrido completo de la tabla.

Se utiliza 2 FAST FULL SCAN para acceder a Reserva_idReserva de la tabla ReservaServicio, es decir, un escaneo de una tabla mas eficiente

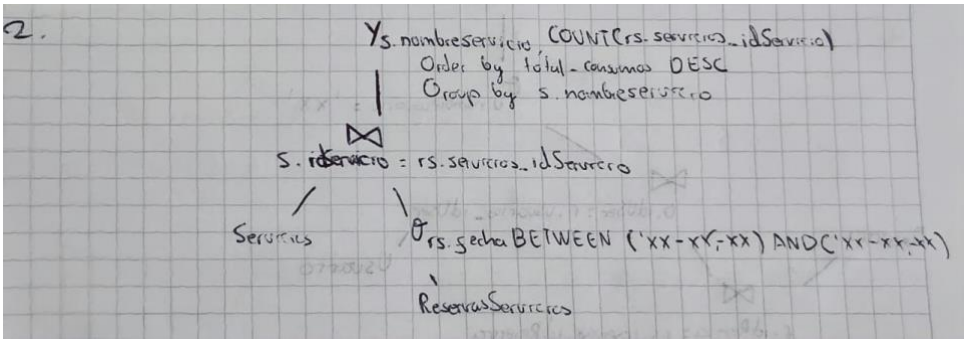
Se utiliza un UNIQUE SCAN para recuperar una fila especifica en la tabla reservas

PLANES DE EJECUCIÓN DE CADA REQUERIMIENTO:

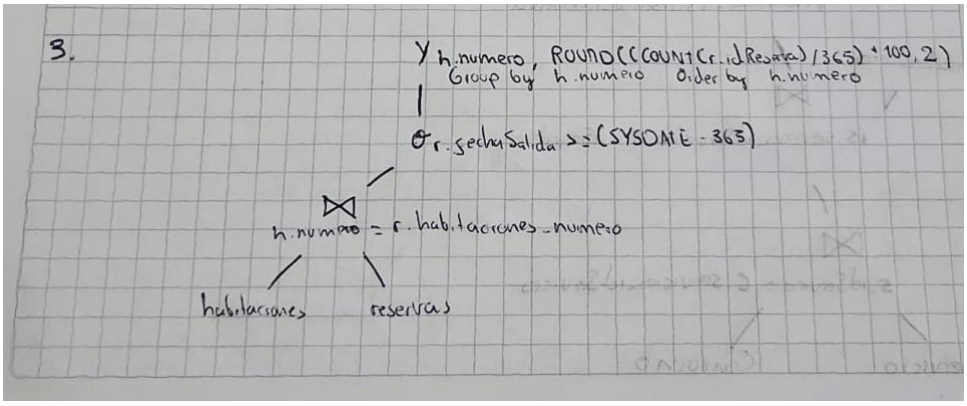
RF1:


1.
γ h.numero, s.nombreservicio, SUM(s.costo)
GROUP BY h.numero, s.nombreservicio
ORDER BY h.numero, s.nombreservicio
σ r.fechaentrada >= TRUNC(SYSDate) - Interval 1 YEAR
⋈
h.numero = r.hab_numero
Habitaciones
⋈
r.idReserva = rss.reserva_idReserva
Reserva
⋈
s.idServicio = rss.servicios_idservicio
ReservasServicios
Servicios

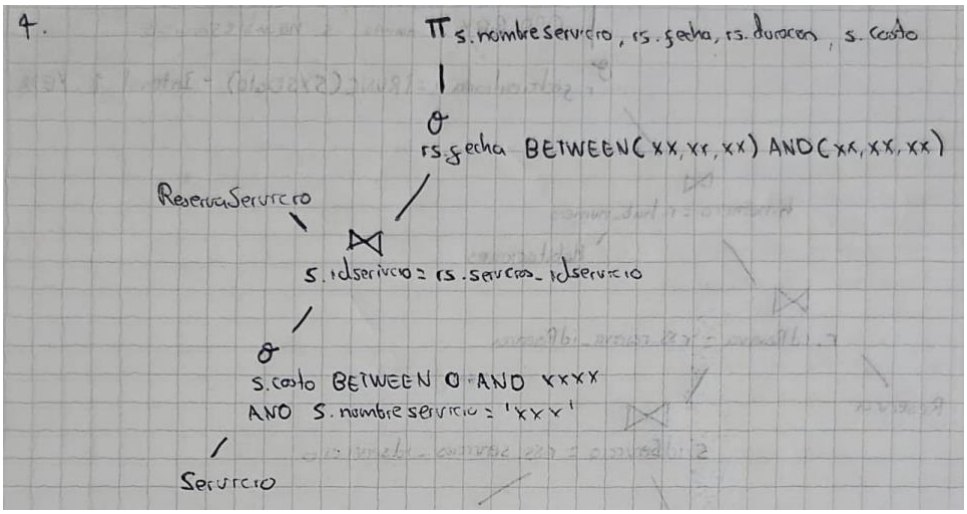
RF2:


2.
γ s.nombreservicio, COUNT(rs.servicios_idServicio)
Order by total_consumos DESC
Group by s.nombreservicio
⋈
s.idservicio = rs.servicios_idServicio
Servicios
σ rs.fecha BETWEEN ('xx-xx-xx') AND ('xx-xx-xx')
ReservasServicios

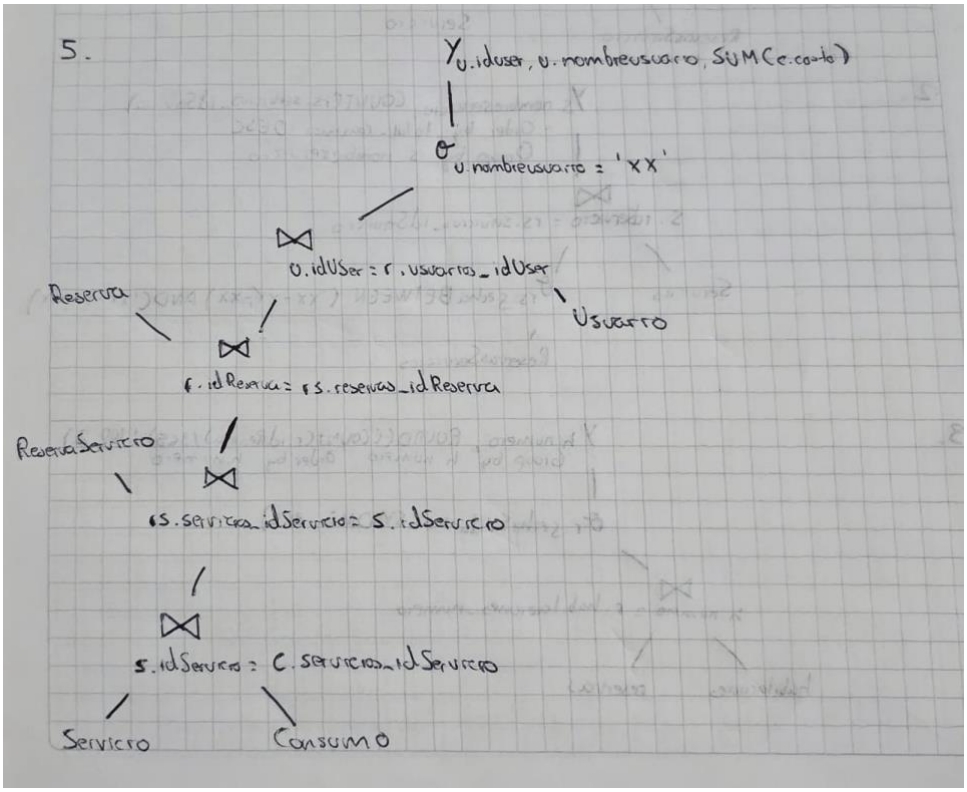
RF3:


3.
γ h.numero, ROUND((COUNT(r.idReserva)/365)*100,2)
Group by h.numero Order by h.numero
σ r.fechaSalida >= (SYSDATE - 365)
⋈
h.numero = r.habitaciones_numero
habitaciones
reservas

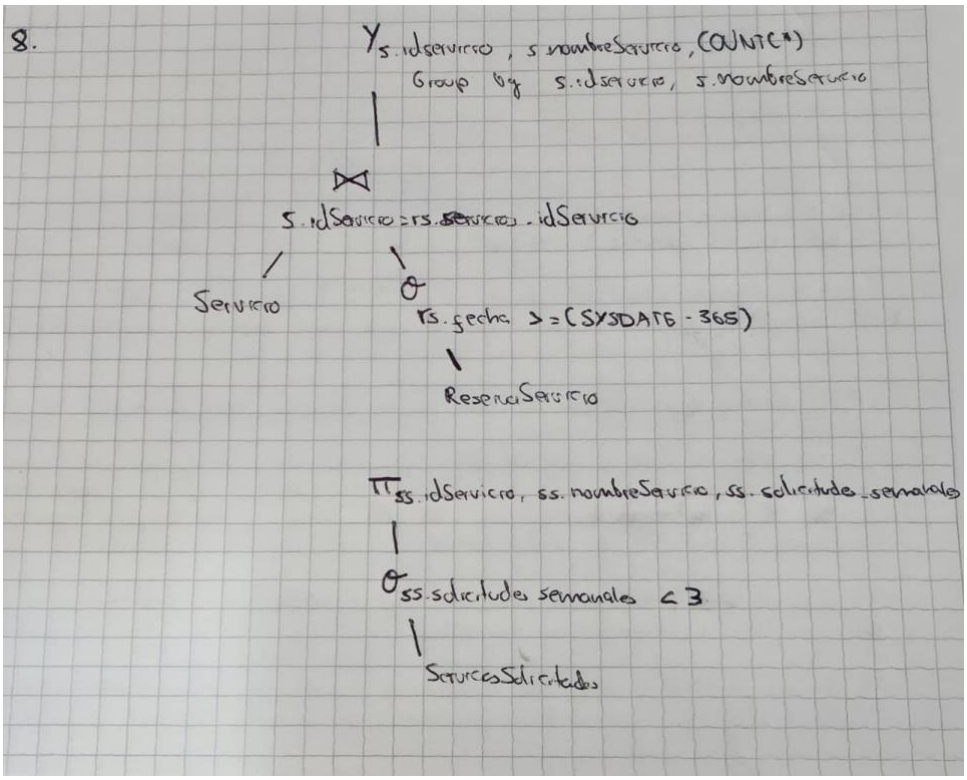
RF4:


4.
π s.nombreservicio, rs.fecha, rs.duracion, s.costo
σ
rs.fecha BETWEEN(xx,xx,xx) AND(xx,xx,xx)
ReservaServicio
⋈
s.idservicio = rs.servicios_idservicio
σ
s.costo BETWEEN 0 AND xxxx
AND s.nombreservicio = 'xxx'
Servicio

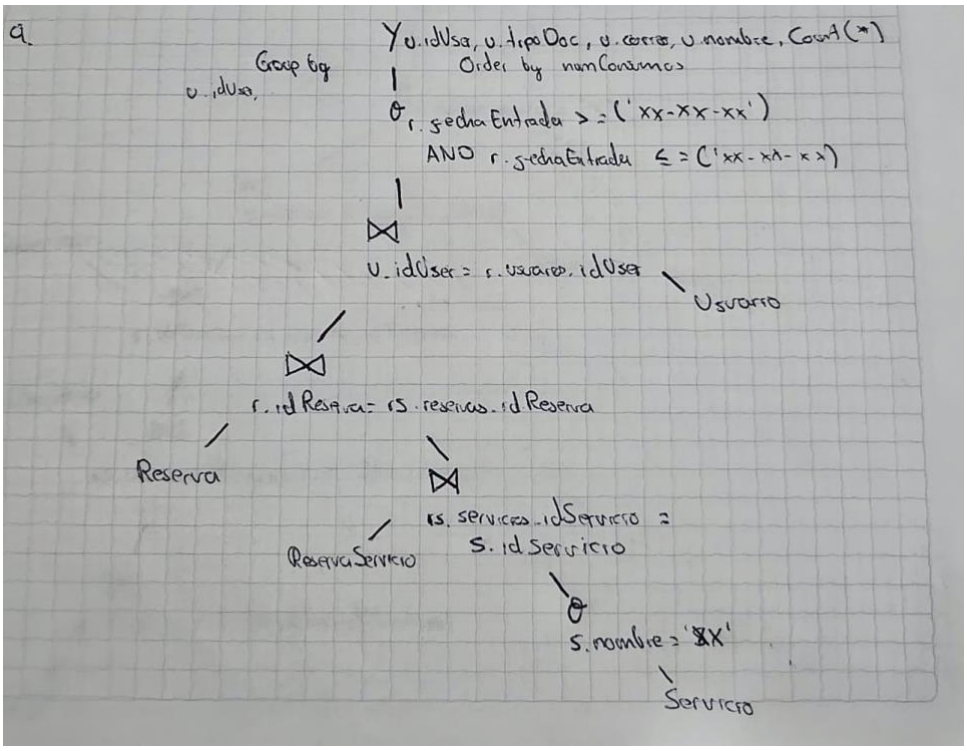
RF5:


5.
γ u.iduser, u.nombreusuario, SUM(c.costo)
σ
u.nombreusuario = 'xx'
⋈
u.idUser = r.usuarios_idUser
Reserva
Usuario
⋈
r.idReserva = rs.reservas_idReserva
ReservaServicio
⋈
rs.servicios_idServicio = s.idServicio
⋈
s.idServicio = c.servicios_idServicio
Servicio
Consumo

RF8:


8.
γ s.idServicio, s.nombreServicio, COUNT(*)
Group by s.idServicio, s.nombreServicio
S.idServicio = rs.servicios.idServicio
Servicio
σ
rs.fecha >= (SYSDATE - 365)
ReservaServicio
π ss.idServicio, ss.nombreServicio, ss.solicitudes_semanales
σ ss.solicitudes semanales < 3
ServiciosSolicitados

RF9:


a.
γ u.idUsa, u.tipoDoc, u.correo, u.nombre, Count(*)
Group by u.idUsa,
Order by numConsumos
σ r.fechaEntrada >= ('xx-xx-xx')
AND r.fechaEntrada <= ('xx-xx-xx')
U.idUser = r.usuario.idUser
Usuario
r.idReserva = rs.reservas.idReserva
Reserva
rs.servicios.idServicio =
S.idServicio
ReservaServicio
σ
S.nombre = '%X'
Servicio