Iteración 4: Sistemas Transaccionales

Juan Esteban Coronel Yela - 202111207
Juan Pablo Martínez Pineda - 202012623

Contenido

1 Diseño Físico

Para el caso de estudio AlohAndes, se es asignada la tarea de optimizar las transacciones y los tiempo de respuesta de los requisitos para así poder hacer consultas mas rápidas, eficientes y en la medida de lo posible menos costosas. Con base a lo anterior, es necesario proponer un nuevo diseño físico que habilite la posibilidad de crear índices para poder acelerar estas consultas, claro está, teniendo en cuenta el tipo de índice y si es necesario para cada requerimiento. De esta manera, a continuación, se presentarán dichos índices con la documentación necesaria para evidenciar su correcto funcionamiento y efectividad.

1.1 Documentación diseño físico

1.1.1 Selección de índices

Para optimizar los Query que manejan los requisitos funcionales, es necesario primero proponer el tipo de índices a utilizar para que la respuesta a estas consultas sea efectiva. Además, se debe tener en cuenta que cada tipo de índice tiene un costo asociado, así que la indexación no debe ser tomado a la ligera. Con base a lo anterior, a continuación, se proponen los índices que ayudaran a optimizar los tiempos asociados los requisitos funcionales.

1. **Índice Reserva.id (Hashing):** Debido a que la mayoría de requisitos funcionales hacen uso de las reservas como núcleo del Query, es necesario optimizar la búsqueda de estos valores. Ahora bien, debido a que reserva tiene una PK en id, podemos usar un índice primario, por ende, podemos usar indexación por Hasheo. Sin embargo, también se podría usar B+, claro está, pero Hashing es sumamente optimo cuando necesitamos hacer muchas comparaciones simples, como, por ejemplo, encontrar una id de una reserva donde id=hash(id). Para el RF4, RF5, RF7 y RF8, los cuales usan una consulta de comparación de reservas, mediante un Query que o bien compara el id con la tabla o con otras tablas, se verían sumamente beneficiado puesto todos estos requerimientos optimizarían el tiempo de traer estas tuplas desde la base de datos. Claro está, usar hashing implica que las tablas se tengan que

organizar según su código de hash en memoria y por ende usar mas recursos del sistema, no obstante, estamos dispuestos a pagar este costo con fin de tener una eficiencia mayor.

2. **Índice Alojamiento.id_alojamiento (Hashing):** De igual manera que con el índice de id para la tabla Reserva, aquí sucede el mismo escenario. Es decir, tenemos una PK en el id del alojamiento de interés que es un índice primario, lo cual nos permite usar hashing sobre este índice. Al tener que hacer muchas comparaciones con el id de un alojamiento cualquiera, se puede encontrar fácilmente al saber su hash con la tabla de hash que hemos creado para el índice. Además, los requisitos funcionales RF9, RF10, RF6 y RF1, que necesitan siempre estar trayendo ciertos valores de la tabla de alojamientos para poder desarrollar sus Query's complejas, tener el hash del índice nos garantiza una optimización efectiva para reducir los tiempos de respuesta de dichos requerimientos. Finalmente, debido a que este es nuestro segundo índice, por el momento es posible considerar el costo de usar dos índices de hashing ya que no se vera muy afectada la memoria disponible que maneja el esquema y así tendremos una alta efectividad de respuesta.

3. **Índice Servicios.nombre (B+):** Hay requisitos funcionales que necesitan considerar las preferencias del cliente, por ejemplo, al registrar una reserva que solicita al cliente que servicios le gustaría incluir en su reserva para validar si hay algún alojamiento que cumpla con dichas condiciones. Para lo anterior, se usa la tabla de servicios la cual tiene el nombre del servicio, que es como un 'alias' único para hacer referencia al servicio. De esta manera, es mucho mas sencillo tener un índice en el nombre del servicio para buscar con agilidad los alojamientos que tengan como atributo ese servicio en la tupla. Así pues, los requisitos que involucren crear reservas considerando los servicios que gustaría tener el cliente, se verían sumamente beneficiados. Cabe recalcar, que el nombre del servicio no es una llave primaria, por ende, sería un índice secundario y no se puede usar Hashing. Lo anterior, indica que el algoritmo más optimo del índice seria usar un Árbol B+, que es sumamente útil para consultas de igualdad de este tipo, e incluso, no es tan costo como él Hash. Finalmente, es importante considerar que como el alias es único, hay una selectividad muy buena y por ende es una buena idea crear un índice sobre este atributo.

4. **Índice Reserva.fecha_llegada y Reserva.fecha_salida (B+):** En muchos requisitos se necesita saber que reservas están activas en que momento, o que reservas fueron registradas para un cierto rango de fechas. Cuando hablamos de comparaciones entre rangos, los arboles B+ son sumamente efectivos. La selectividad de estos atributos no es tan alta como los anteriormente presentados, pero debido a la gran variedad de fechas en las reservas de los clientes, se puede hacer un filtrado de un pase sobre la tabla de reserva en llegada, y después comparar que la fecha salida corresponda con lo que se esta buscando. De esta manera, ahora obtener estos rangos para hacer análisis o Query's en los requerimientos, es mucho mas sencillo y eficiente.

5. **Índice Usuario.cedula (Hashing):** En casi todos los requerimientos, tanto funcionales como de consulta, siempre se necesita hacer joins o comparaciones con la tabla de usuarios para traer sus estadísticas. En respuesta a lo anterior, es muy importante tener un índice en la cedula de usuario, la cual no solamente tiene una alta selectividad puesto es un valor único, si no también, es una llave primaria, lo cual nos da un índice primario y por ende somos capaces de usar Hashing. Como se ha explicado anteriormente, hashing es sumamente útil cuando hacemos comparaciones, que, en este caso, la mayoría buscaran un tipo de cedula para el cliente en la tabla de usuarios, trayendo así los valores de la tupla con la información de la persona de interés.

1.1.2 Índices creados por Oracle (default)

INDEX_NAME	TABLE_NAME	COLUMN_NAME
BIN$+h8dMufVM8ngVWEJKLg5TQ==$C	BIN$+h8dMufWM8ngVWEJKLg5TQ==$C	IDRESERVA
BIN$+hSzFvX2AbDgVWEJKLg5TQ==$C	BIN$+hSzFvX4AbDgVWEJKLg5TQ==$C	NOMBRE
BIN$+hSzFvX3AbDgVWEJKLg5TQ==$C	BIN$+hSzFvX4AbDgVWEJKLg5TQ==$C	ID
BIN$+hSzFvX8AbDgVWEJKLg5TQ==$C	BIN$+hSzFvX9AbDgVWEJKLg5TQ==$C	ID
BIN$+hSzFvXkAbDgVWEJKLg5TQ==$C	BIN$+hSzFvXlAbDgVWEJKLg5TQ==$C	ID_BAR
BIN$+hSzFvXkAbDgVWEJKLg5TQ==$C	BIN$+hSzFvXlAbDgVWEJKLg5TQ==$C	ID_BEBIDA
BIN$+hSzFvXkAbDgVWEJKLg5TQ==$C	BIN$+hSzFvXlAbDgVWEJKLg5TQ==$C	HORARIO
BIN$+hSzFvXoAbDgVWEJKLg5TQ==$C	BIN$+hSzFvXqAbDgVWEJKLg5TQ==$C	ID_BEBEDOR
BIN$+hSzFvXpAbDgVWEJKLg5TQ==$C	BIN$+hSzFvXqAbDgVWEJKLg5TQ==$C	ID_BAR
BIN$+hSzFvXpAbDgVWEJKLg5TQ==$C	BIN$+hSzFvXqAbDgVWEJKLg5TQ==$C	ID_BEBEDOR
BIN$+hSzFvXpAbDgVWEJKLg5TQ==$C	BIN$+hSzFvXqAbDgVWEJKLg5TQ==$C	HORARIO
BIN$+hSzFvXvAbDgVWEJKLg5TQ==$C	BIN$+hSzFvXwAbDgVWEJKLg5TQ==$C	IDSILLA
BIN$+hSzFvXvAbDgVWEJKLg5TQ==$C	BIN$+hSzFvXwAbDgVWEJKLg5TQ==$C	IDFUNCION
BIN$+hSzFvXzAbDgVWEJKLg5TQ==$C	BIN$+hSzFvX0AbDgVWEJKLg5TQ==$C	ID
BIN$+hSzFvYAAbDgVWEJKLg5TQ==$C	BIN$+hSzFvYBAbDgVWEJKLg5TQ==$C	IDRESERVA
BIN$+hSzFvYDAbDgVWEJKLg5TQ==$C	BIN$+hSzFvYEAbDgVWEJKLg5TQ==$C	ID
BIN$+puJ8lAGEH/gVWEJKLg5TQ==$C	BIN$+puJ8lAHEH/gVWEJKLg5TQ==$C	IDRESERVA
BIN$+pvkm1W3EQPgVWEJKLg5TQ==$C	BIN$+pvkm1W4EQPgVWEJKLg5TQ==$C	ID_HOSTAL
BIN$+pvkm1W9EQPgVWEJKLg5TQ==$C	BIN$+pvkm1W+EQPgVWEJKLg5TQ==$C	ID_HOTEL
BIN$+pvkm1WsEQPgVWEJKLg5TQ==$C	BIN$+pvkm1WtEQPgVWEJKLg5TQ==$C	ID_ALOJAMIENTO
BIN$+pvkm1WxEQPgVWEJKLg5TQ==$C	BIN$+pvkm1WyEQPgVWEJKLg5TQ==$C	ID_CONTRATO
BIN$+pvkm1XCEQPgVWEJKLg5TQ==$C	BIN$+pvkm1XDEQPgVWEJKLg5TQ==$C	ID_OPERADOR
BIN$+pvkm1XHEQPgVWEJKLg5TQ==$C	BIN$+pvkm1XIEQPgVWEJKLg5TQ==$C	ID_PARTICULAR
BIN$+pvkm1XMEQPgVWEJKLg5TQ==$C	BIN$+pvkm1XNEQPgVWEJKLg5TQ==$C	ID_PROPUESTA
BIN$+pvkm1XVEQPgVWEJKLg5TQ==$C	BIN$+pvkm1XWEQPgVWEJKLg5TQ==$C	ID_RESERVA
BIN$+pvkm1XbEQPgVWEJKLg5TQ==$C	BIN$+pvkm1XcEQPgVWEJKLg5TQ==$C	ID_RESIDENCIA
BIN$+pvkm1XhEQPgVWEJKLg5TQ==$C	BIN$+pvkm1XiEQPgVWEJKLg5TQ==$C	ID_SERVICIO
BIN$+pvkm1XmEQPgVWEJKLg5TQ==$C	BIN$+pvkm1XnEQPgVWEJKLg5TQ==$C	CEDULA
BIN$+q85t1R+VzzgVWEJKLg5TQ==$C	BIN$+q85t1R/VzzgVWEJKLg5TQ==$C	ID_PROPUESTA
BIN$+q85t1R0VzzgVWEJKLg5TQ==$C	BIN$+q85t1R1VzzgVWEJKLg5TQ==$C	ID_RESERVA
BIN$+q85t1R5VzzgVWEJKLg5TQ==$C	BIN$+q85t1R6VzzgVWEJKLg5TQ==$C	CEDULA
BIN$+q85t1RbVzzgVWEJKLg5TQ==$C	BIN$+q85t1RcVzzgVWEJKLg5TQ==$C	ID_RESERVA
BIN$+q85t1RgVzzgVWEJKLg5TQ==$C	BIN$+q85t1RhVzzgVWEJKLg5TQ==$C	CEDULA
BIN$+q85t1RrVzzgVWEJKLg5TQ==$C	BIN$+q85t1RsVzzgVWEJKLg5TQ==$C	ID_ALOJAMIENTO
BIN$+q85t1SJVzzgVWEJKLg5TQ==$C	BIN$+q85t1SKVzzgVWEJKLg5TQ==$C	ID_ALOJAMIENTO
BIN$+q85t1SOVzzgVWEJKLg5TQ==$C	BIN$+q85t1SPVzzgVWEJKLg5TQ==$C	ID_OPERADOR
BIN$+qpQ1mQ0BcHgVWEJKLg5TQ==$C	BIN$+qpQ1mQ1BcHgVWEJKLg5TQ==$C	ID_CONTRATO
BIN$+qpQ1mQ6BcHgVWEJKLg5TQ==$C	BIN$+qpQ1mQ7BcHgVWEJKLg5TQ==$C	ID_HOSTAL

Tabla 1

INDEX_NAME	TABLE_NAME	COLUMN_NAME
BIN$+qpQ1mR0BcHgVWEJKLg5TQ==$C	BIN$+qpQ1mR1BcHgVWEJKLg5TQ==$C	ID_OPERADOR
BIN$+qpQ1mR5BcHgVWEJKLg5TQ==$C	BIN$+qpQ1mR6BcHgVWEJKLg5TQ==$C	ID_OPERADOR
BIN$+qpQ1mRABcHgVWEJKLg5TQ==$C	BIN$+qpQ1mRBBcHgVWEJKLg5TQ==$C	ID_HOTEL
BIN$+qpQ1mRFBcHgVWEJKLg5TQ==$C	BIN$+qpQ1mRGBcHgVWEJKLg5TQ==$C	ID_PARTICULAR
BIN$+qpQ1mRKBcHgVWEJKLg5TQ==$C	BIN$+qpQ1mRLBcHgVWEJKLg5TQ==$C	ID_PROPUESTA
BIN$+qpQ1mRTBcHgVWEJKLg5TQ==$C	BIN$+qpQ1mRUBcHgVWEJKLg5TQ==$C	ID_RESERVA
BIN$+qpQ1mRZBcHgVWEJKLg5TQ==$C	BIN$+qpQ1mRaBcHgVWEJKLg5TQ==$C	ID_RESIDENCIA
BIN$+qpQ1mRfBcHgVWEJKLg5TQ==$C	BIN$+qpQ1mRgBcHgVWEJKLg5TQ==$C	ID_SERVICIO
BIN$+qpQ1mRkBcHgVWEJKLg5TQ==$C	BIN$+qpQ1mRlBcHgVWEJKLg5TQ==$C	CEDULA
BIN$+qpQ1mRvBcHgVWEJKLg5TQ==$C	BIN$+qpQ1mRwBcHgVWEJKLg5TQ==$C	ID_ALOJAMIENTO
BIN$9sfefvcWUwzgVWEJKLg5TQ==$C	BIN$9sfefvcXUwzgVWEJKLg5TQ==$C	ID
BIN$9zfp3QsCY9LgVWEJKLg5TQ==$C	BIN$9zfp3QsDY9LgVWEJKLg5TQ==$C	ID
SYS_C00996285	A_OPERADORES	ID_OPERADOR
SYS_C00996294	A_ALOJAMIENTOS	ID_ALOJAMIENTO
SYS_C00996298	A_USUARIOS	CEDULA
SYS_C00996305	A_RESERVAS	ID_RESERVA
SYS_C00996310	A_PROPUESTAS	ID_PROPUESTA
SYS_C00996769	A_SERVICIOS	ID_SERVICIO
SYS_C00997052	A_ALOJDESHABILITADOS	ID_ALOJAMIENTO
SYS_C00998358	A_CONTRATOS	ID_CONTRATO
SYS_C00998362	A_PARTICULARES	ID_PARTICULAR
SYS_C00998367	A_RESIDENCIASU	ID_RESIDENCIA
SYS_C00998372	A_HOTELES	ID_HOTEL
SYS_C00998377	A_HOSTALES	ID_HOSTAL

Tabla 2

Como es evidente, Oracle ® ha creado índices sobre todas las llaves primarias de todas las tablas. Es decir, sobre todos los id, que para este esquema son las llaves primarias que identifican cada tabla unívocamente, Oracle tomo la decisión de usar índices en estos valores para así acceder a las tablas de manera mas eficiente. De esta manera, el sistema manejador de base de datos ya tiene las indexaciones que pueden llevar a cualquier plan de ejecución hacia las tablas que necesite, que, en este caso, apuntaran siempre a INDICES PRIMARIOS, que contienen las tuplas referenciadas. El comando SQL para traer la información mostrada en la *Tabla 1* y *Tabla 2* es el siguiente:

```
SELECT index_name, table_name, column_name
        FROM all_ind_columns
        WHERE index_owner = 'ISIS2304D06202310';
```

1.2 Documentación del análisis realizado para cada RFC

1.2.1 Documentación del escenario de pruebas

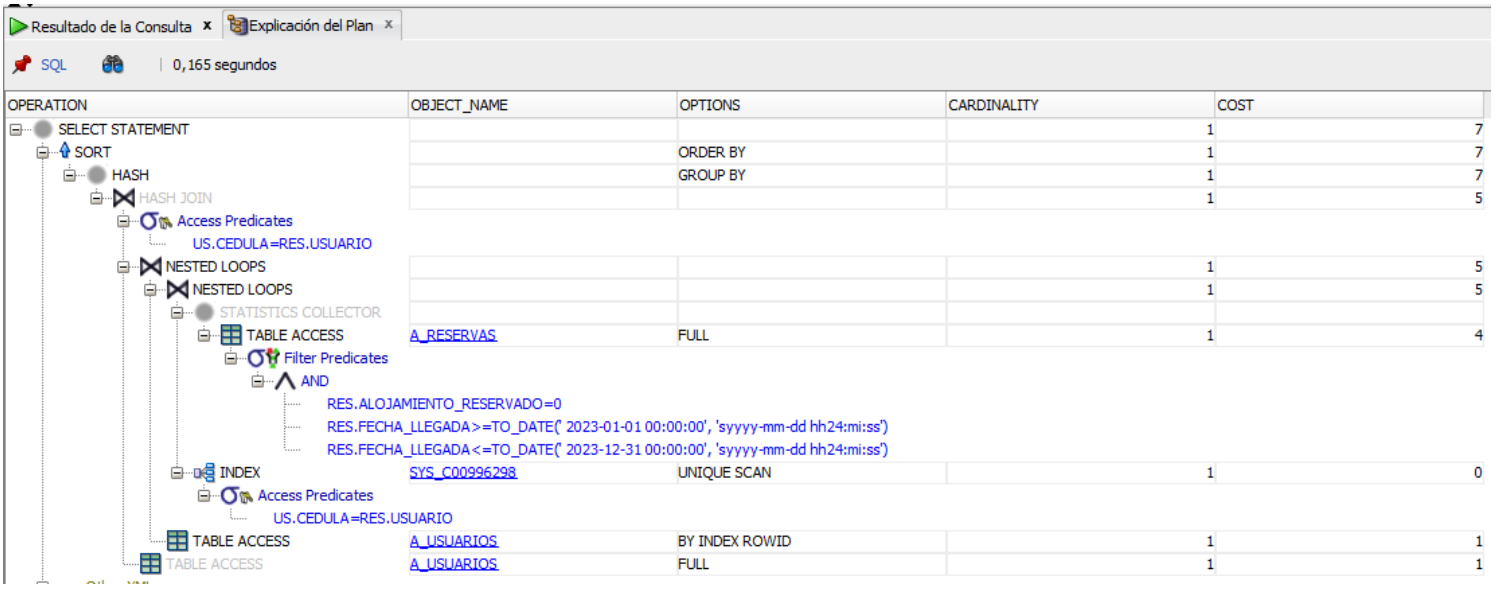
1.2.1.1 RF10:

Comando SQL para lograr la consulta según sea el caso:

```
--RF10: En caso de que se quiera agrupar por información del usuario--
-- parámetros: id alojamiento, fecha.
--
SELECT us.*, COUNT(aloj.id_alojamiento) as cantidad_reservas_usuario
FROM A_USUARIOS us INNER JOIN A_RESERVAS res
ON us.cedula = res.usuario INNER JOIN A_ALOJAMIENTOS aloj on
res.alojamiento_reservado = aloj.id_alojamiento
WHERE res.alojamiento_reservado = 0
AND res.fecha_llegada BETWEEN TO_DATE('2023-01-01', 'YYYY-MM-DD') AND
TO_DATE('2023-12-31', 'YYYY-MM-DD')

GROUP BY us.nombre, us.cedula, us.relacion_universidad
ORDER BY us.nombre, us.cedula, us.relacion_universidad
;

--RF10: En caso de que se quiera agrupar por oferta de alojamiento--
-- parametros: id alojamiento, fecha.
--
SELECT aloj.id_alojamiento as alojamiento, COUNT(distinct us.nombre) as
cantidad_reservas_usuario
FROM A_USUARIOS us INNER JOIN A_RESERVAS res
ON us.cedula = res.usuario INNER JOIN A_ALOJAMIENTOS aloj on
res.alojamiento_reservado = aloj.id_alojamiento
WHERE res.alojamiento_reservado = 0
AND res.fecha_llegada BETWEEN TO_DATE('2023-01-01', 'YYYY-MM-DD') AND
TO_DATE('2023-12-31', 'YYYY-MM-DD')

GROUP BY aloj.id_alojamiento
ORDER BY aloj.id_alojamiento
;

--RF10: En caso de que se quiera agrupar por tipo de alojamiento--
-- parametros: id alojamiento, fecha.
--
SELECT aloj.nombre_alojamiento as tipo_alojamiento, COUNT(distinct us.nombre)
as cantidad_reservas_usuario
FROM A_USUARIOS us INNER JOIN A_RESERVAS res
ON us.cedula = res.usuario INNER JOIN A_ALOJAMIENTOS aloj on
res.alojamiento_reservado = aloj.id_alojamiento
WHERE res.alojamiento_reservado = 0
AND res.fecha_llegada BETWEEN TO_DATE('2023-01-01', 'YYYY-MM-DD') AND
TO_DATE('2023-12-31', 'YYYY-MM-DD')
```

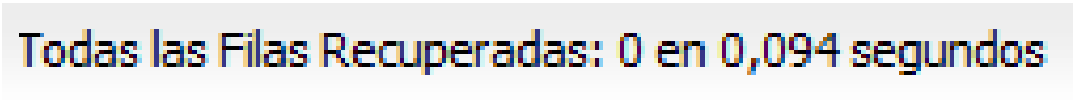
```
GROUP BY aloj.nombre_alojamiento
ORDER BY aloj.nombre_alojamiento
;
```

Plan de ejecución de la consulta según Oracle:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	7
SORT		ORDER BY	1	7
HASH		GROUP BY	1	7
HASH JOIN			1	5
Access Predicates				
US.CEDULA=RES.USUARIO				
NESTED LOOPS			1	5
NESTED LOOPS			1	5
STATISTICS COLLECTOR				
TABLE ACCESS	A_RESERVAS	FULL	1	4
Filter Predicates				
AND				
RES.ALOJAMIENTO_RESERVADO=0				
RES.FECHA_LLEGADA>=TO_DATE(' 2023-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				
RES.FECHA_LLEGADA<=TO_DATE(' 2023-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				
INDEX	SYS_C00996298	UNIQUE SCAN	1	0
Access Predicates				
US.CEDULA=RES.USUARIO				
TABLE ACCESS	A_USUARIOS	BY INDEX ROWID	1	1
TABLE ACCESS	A_USUARIOS	FULL	1	1

Tiempo de ejecución de la consulta por default:



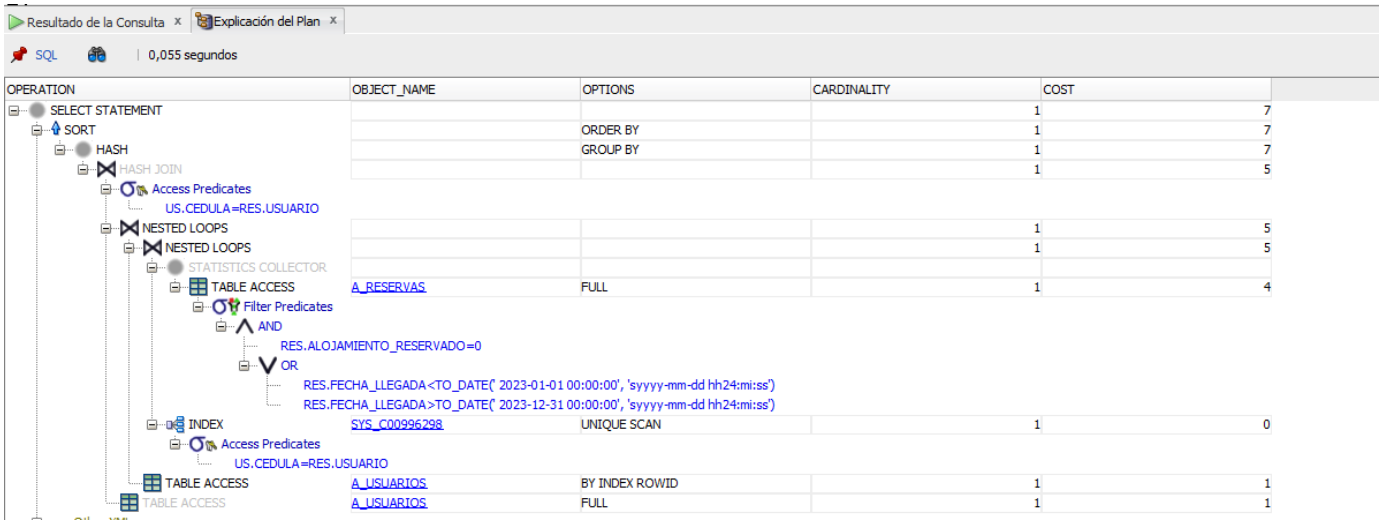
1.2.1.2 RF11:

Comando SQL para lograr la consulta según sea el caso:

```
--RF11------------------------------------------>

--RF11: En caso de que se quiera agrupar por informacion del usuario--
-- parametros: id alojamiento, fecha.
--
SELECT us.*, COUNT(aloj.id_alojamiento) as cantidad_reservas_usuario
FROM A_USUARIOS us INNER JOIN A_RESERVAS res
ON us.cedula = res.usuario INNER JOIN A_ALOJAMIENTOS aloj on
res.alojamiento_reservado = aloj.id_alojamiento
WHERE res.alojamiento_reservado = 0
AND res.fecha_llegada NOT BETWEEN TO_DATE('2023-01-01', 'YYYY-MM-DD') AND
TO_DATE('2023-12-31', 'YYYY-MM-DD')

GROUP BY us.nombre, us.cedula, us.relacion_universidad
ORDER BY us.nombre, us.cedula, us.relacion_universidad
```

```
;

--RF11: En caso de que se quiera agrupar por oferta de alojamiento--
-- parametros: id alojamiento, fecha.
--
SELECT aloj.id_alojamiento as alojamiento, COUNT(distinct us.nombre) as
cantidad_reservas_usuario
FROM A_USUARIOS us INNER JOIN A_RESERVAS res
ON us.cedula = res.usuario INNER JOIN A_ALOJAMIENTOS aloj on
res.alojamiento_reservado = aloj.id_alojamiento
WHERE res.alojamiento_reservado = 0
AND res.fecha_llegada NOT BETWEEN TO_DATE('2023-01-01', 'YYYY-MM-DD') AND
TO_DATE('2023-12-31', 'YYYY-MM-DD')

GROUP BY aloj.id_alojamiento
ORDER BY aloj.id_alojamiento
;

--RF11: En caso de que se quiera agrupar por tipo de alojamiento--
-- parametros: id alojamiento, fecha.
--
SELECT aloj.nombre_alojamiento as tipo_alojamiento, COUNT(distinct us.nombre)
as cantidad_reservas_usuario
FROM A_USUARIOS us INNER JOIN A_RESERVAS res
ON us.cedula = res.usuario INNER JOIN A_ALOJAMIENTOS aloj on
res.alojamiento_reservado = aloj.id_alojamiento
WHERE res.alojamiento_reservado = 0
AND res.fecha_llegada NOT BETWEEN TO_DATE('2023-01-01', 'YYYY-MM-DD') AND
TO_DATE('2023-12-31', 'YYYY-MM-DD')

GROUP BY aloj.nombre_alojamiento
ORDER BY aloj.nombre_alojamiento;
```

Plan de ejecución de la consulta según Oracle:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	7
SORT		ORDER BY	1	7
HASH		GROUP BY	1	7
HASH JOIN			1	5
Access Predicates				
US.CEDULA=RES.USUARIO				
NESTED LOOPS			1	5
NESTED LOOPS			1	5
STATISTICS COLLECTOR				
TABLE ACCESS	A_RESERVAS	FULL	1	4
Filter Predicates				
AND				
RES.ALOJAMIENTO_RESERVADO=0				
OR				
RES.FECHA_LLEGADA<TO_DATE(' 2023-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				
RES.FECHA_LLEGADA>TO_DATE(' 2023-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss')				
INDEX	SYS_C00996298	UNIQUE SCAN	1	0
Access Predicates				
US.CEDULA=RES.USUARIO				
TABLE ACCESS	A_USUARIOS	BY INDEX ROWID	1	1
TABLE ACCESS	A_USUARIOS	FULL	1	1

Tiempo de ejecución de la consulta por default:

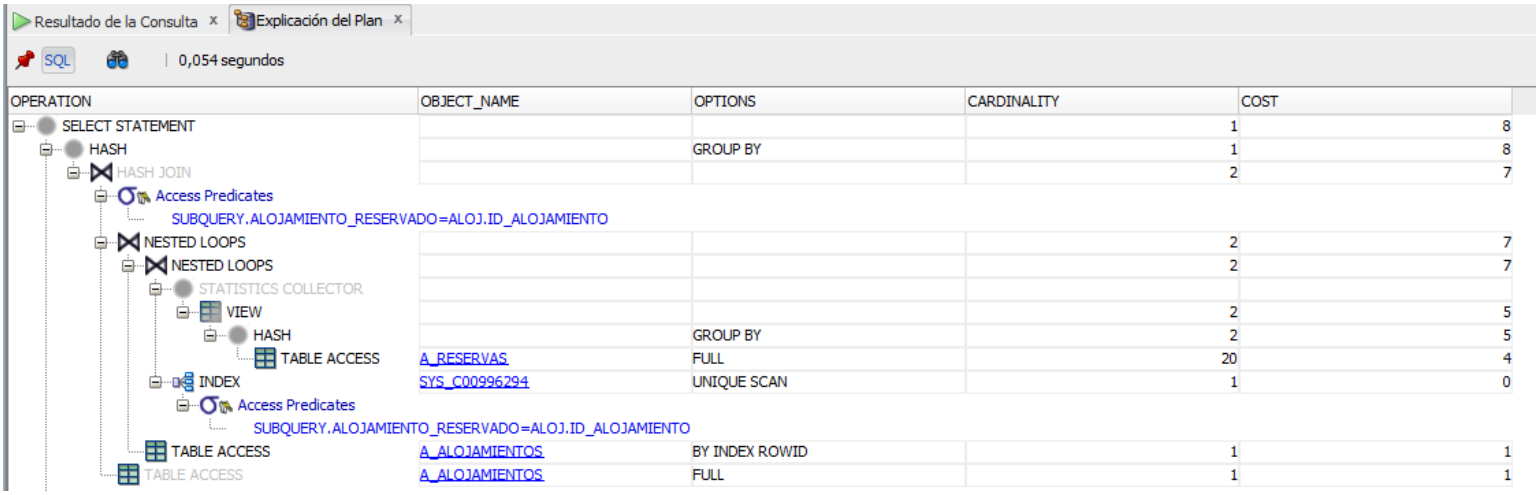
1.2.1.3 RF12:

Comando SQL para lograr la consulta según sea el caso:

```
--RF12----------------------------->

--Recibe como parametro la semana a evaluar

--RF12:ALOJAMIENTO DE MINIMA OCUPACION:
SELECT aloj.nombre_alojamiento AS Alojamiento, MIN(total_reservas) AS
Minimo_Reservas
FROM (
    SELECT res.alojamiento_reservado, COUNT(*) AS total_reservas
    FROM A_ALOJAMIENTOS aloj
    INNER JOIN A_RESERVAS res ON aloj.id_alojamiento =
res.alojamiento_reservado
    GROUP BY res.alojamiento_reservado
) subquery
INNER JOIN A_ALOJAMIENTOS aloj ON subquery.alojamiento_reservado =
aloj.id_alojamiento
GROUP BY aloj.nombre_alojamiento;

--RF12:OPERADORES MAS SOLICITADOS
SELECT op.nombre AS Operador, COUNT(*) AS TotalReservas
FROM A_OPERADORES op
INNER JOIN A_ALOJAMIENTOS aloj ON op.id_operador = aloj.operador
INNER JOIN A_RESERVAS res ON aloj.id_alojamiento = res.alojamiento_reservado
GROUP BY op.nombre
ORDER BY COUNT(*) DESC;


--RF12: OPERADORES MENOS SOLICITADOS
```

```sql
SELECT op.nombre AS Operador, COUNT(*) AS TotalReservas
FROM A_OPERADORES op
INNER JOIN A_ALOJAMIENTOS aloj ON op.id_operador = aloj.operador
INNER JOIN A_RESERVAS res ON aloj.id_alojamiento = res.alojamiento_reservado
GROUP BY op.nombre
ORDER BY COUNT(*) ASC;
```

Plan de ejecución de la consulta según Oracle:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	8
HASH		GROUP BY	1	8
HASH JOIN			2	7
Access Predicates				
SUBQUERY.ALOJAMIENTO_RESERVADO=ALOJ.ID_ALOJAMIENTO				
NESTED LOOPS			2	7
NESTED LOOPS			2	7
STATISTICS COLLECTOR				
VIEW			2	5
HASH		GROUP BY	2	5
TABLE ACCESS	A_RESERVAS	FULL	20	4
INDEX	SYS_C00996294	UNIQUE SCAN	1	0
Access Predicates				
SUBQUERY.ALOJAMIENTO_RESERVADO=ALOJ.ID_ALOJAMIENTO				
TABLE ACCESS	A_ALOJAMIENTOS	BY INDEX ROWID	1	1
TABLE ACCESS	A_ALOJAMIENTOS	FULL	1	1

Tiempo de ejecución de la consulta por default:

Todas las Filas Recuperadas: 2 en 0,078 segundos

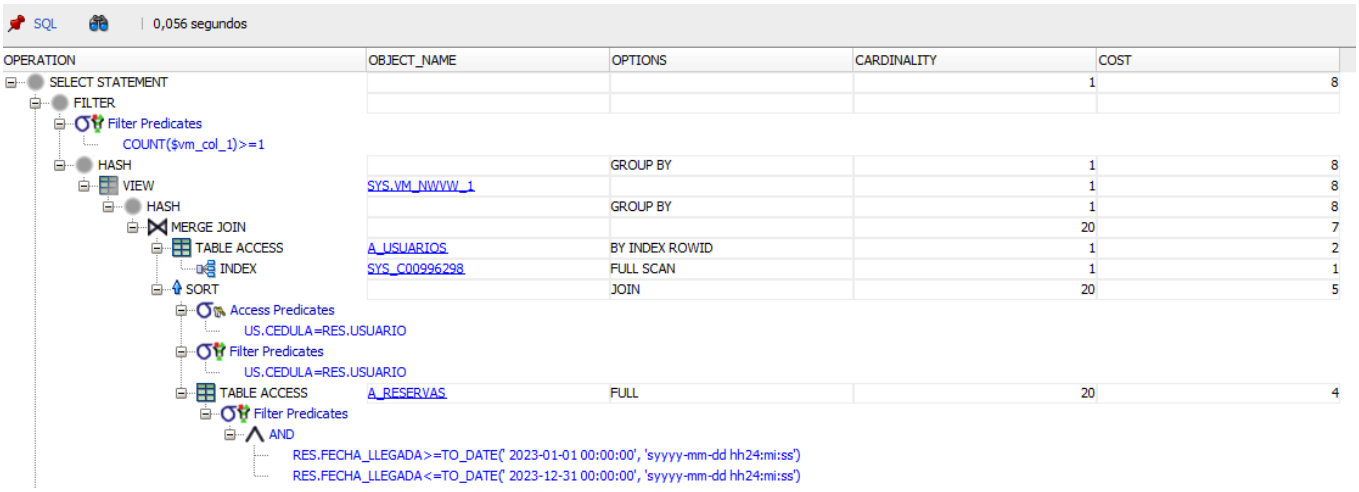
1.2.1.4 RF13:

Comando SQL para lograr la consulta según sea el caso:

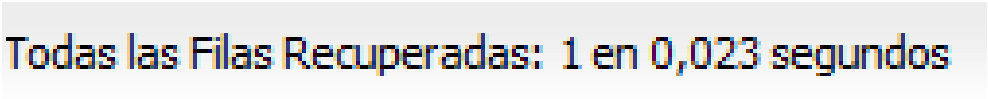
```sql
--RF13---------------------------------------->


--RF13: cliente bueno 1: Hacen al menos 1 reserva en el mes
-- Se recibe como parametro
-- la fecha del mes
SELECT us.*, COUNT(DISTINCT res.fecha_llegada) as
justificacion_buen_cliente_numero_reservas
FROM A_USUARIOS us
INNER JOIN A_RESERVAS res ON us.cedula = res.usuario
WHERE res.fecha_llegada >= TO_DATE('2023-01-01', 'YYYY-MM-DD')
  AND res.fecha_llegada <= TO_DATE('2023-12-31', 'YYYY-MM-DD')
GROUP BY us.cedula, us.nombre, us.relacion_universidad
HAVING COUNT(DISTINCT res.fecha_llegada) >= 1;
```

```sql
--RF13: cliente bueno 2: Reservan siempre alojamientos costosos
SELECT us.*, (SELECT COUNT(*)
              FROM A_RESERVAS res
              INNER JOIN A_ALOJAMIENTOS aloj ON res.alojamiento_reservado =
aloj.id_alojamiento
              WHERE us.cedula = res.usuario AND aloj.precio > 150) AS
justificacion_buen_cliente_cantidad_reservas_mayor_150
FROM A_USUARIOS us
WHERE NOT EXISTS (
    SELECT 1
    FROM A_RESERVAS res
    INNER JOIN A_ALOJAMIENTOS aloj ON res.alojamiento_reservado =
aloj.id_alojamiento
    WHERE us.cedula = res.usuario AND aloj.precio <= 150
);


--RF13: cliente bueno 3: Reservan siempre alojamientos tipo suite
SELECT us.*, (SELECT COUNT(*)
              FROM A_RESERVAS res
              INNER JOIN A_ALOJAMIENTOS aloj ON res.alojamiento_reservado =
aloj.id_alojamiento
              WHERE us.cedula = res.usuario AND aloj.nombre_alojamiento =
'suite') AS justificacion_buen_cliente_cantidad_reservas_suite
FROM A_USUARIOS us
WHERE NOT EXISTS (
    SELECT 1
    FROM A_RESERVAS res
    INNER JOIN A_ALOJAMIENTOS aloj ON res.alojamiento_reservado =
aloj.id_alojamiento
    WHERE us.cedula = res.usuario AND aloj.nombre_alojamiento <> 'suite'
);
```

Plan de ejecución de la consulta según Oracle:



Tiempo de ejecución de la consulta por default:



1.2.2 Planes de ejecución propuestos

[FALTA POR HACER]

2 Población de un millón de datos

La obtención de los datos para la carga masiva se realizó con ayuda de scripts en Python realizados con el módulo Random de Python para la generación de números pseudo-aleatorios y el paquete Faker para la generación de nombres de ejemplo para poblar las tablas.

Se realizaron 3 casos para la población de datos, 5% de los datos (correspondiente a un total de 50.000 registros en la base de datos), 20% de los datos (correspondiente a un total de 200.000 registros en la base de datos) y el 100% de los datos (correspondiente a un total de 1.000.000 de registros en la base de datos). Todos los archivos de datos generados se encuentran en el repositorio en la carpeta *data* (*Carga_5pct.rar*, *Carga_20pct.rar* y *Carga_100pct.rar*).

Asimismo, los scripts de Python también se encuentran en el repositorio en esa misma carpeta (*PythonScripts.rar*). A continuación, se muestran los fragmentos de código encargados de esto:

Generar_operadores.py

```
with open('gen_operadores_20pct.sql', 'w') as file:
    x = 1
    for i in range(len(nombres)):
        for j in range(len(apellidos)):
            values = [x, f'\'{nombres[i]} {apellidos[j]}\'']
            formatted_values = ', '.join(str(value) for value in values)
            file.write(f"INSERT INTO {table_name} ({', '.join(column_names)}) VALUES ({formatted_values});\n")
            x += 1
```

Generar_alojamientos.py

```
with open('gen_alojamientos_1pct.sql', 'w') as file:
    for i in range(1, num_inserts+1):
        a_name = faker.street_name()
        a_name = re.sub(r"([^n\u0300-\u036f]|n(?!\u0303(?![\u0300-\u036f])))[\u0300-\u036f]+", r"\1", normalize( "NFD", a_name), 0, re.I)
        a_name = normalize('NFC',a_name)
        a_name = a_name.replace('ñ','n')
        tipo = random.choice(['Hotel', 'Hostal', 'Habitacion'])
        values = [i, random.randint(1, 200), random.randint(1, 200), random.randint(20000, 200000), random.choice(['\'10AM - 10PM\'', '\'6AM - 6P

        formatted_values = ', '.join(str(value) for value in values)

        file.write(f"INSERT INTO {table_name} ({', '.join(column_names)}) VALUES ({formatted_values});\n")

        if tipo == 'Hotel':
            file.write(f'INSERT INTO A_Hoteles (id_hotel, nombre_hotel, registro_legal) VALUES ({i}, \'{tipo} {a_name}\', \'FILE...\');\n')
        elif tipo == 'Hostal':
            file.write(f'INSERT INTO A_Hostales (id_hostal, nombre_hostal, registro_legal) VALUES ({i}, \'{tipo} {a_name}\', \'FILE...\');\n')
        else:
            file.write(f'INSERT INTO A_Particulares (id_particular, nombre_particular) VALUES ({i}, \'{tipo} {a_name}\');\n')
```

Generar_usuarios.py

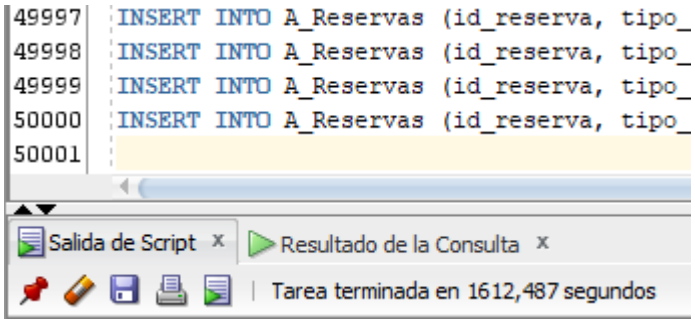
```
with open('gen_usuarios_1pct.sql', 'w') as file:
    for i in range(1000000000, 1000001000):
        u_name = faker.name()
        u_name = re.sub(r"([^n\u0300-\u036f]|n(?!\u0303(?![\u0300-\u036f])))[\u0300-\u036f]+", r"\1", normalize( "NFD", u_name), 0, re.I)
        u_name = normalize('NFC',u_name)
        u_name = u_name.replace('ñ','n')
        grado = random.choice(['Pregrado', 'Especializacion', 'Maestria', 'Doctorado'])
        user = random.choice(['Estudiante', 'Asistente', 'Docente'])

        values = [f'\'{u_name}\'', f'\'{i}\'', f'\'{user} {grado}\'']

        formatted_values = ', '.join(str(value) for value in values)

        file.write(f"INSERT INTO {table_name} ({', '.join(column_names)}) VALUES ({formatted_values});\n")
```

Generar_reservas.py

```
with open('gen_reservas_20pct.sql', 'w') as file:
    for i in range(1, num_inserts+1):
        time_delta = random.randint(1,30)
        fecha_inicio = faker.date_between(start_date='-5y', end_date='today')
        fecha_fin = faker.date_between(start_date=fecha_inicio, end_date=fecha_inicio + datetime.timedelta(days=time_delta))
        fecha_llegada = fecha_inicio.strftime("%d/%m/%Y")
        fecha_salida = fecha_fin.strftime("%d/%m/%Y")

        values = [i, '\'Ninguno\'', f'\'{fecha_llegada}\'', f'\'{fecha_salida}\'', random.randint(10000, 250000), random.randint(1000000000, 1000
        formatted_values = ', '.join(str(value) for value in values)

        file.write(f"INSERT INTO {table_name} ({', '.join(column_names)}) VALUES ({formatted_values});\n")
```

El volumen de datos solicitado fue generado, principalmente, especificando cuántas veces debía ejecutarse el loop *for_in range()* para que el código se encargara de escribir sobre el archivo creado los datos solicitados. Por ejemplo, para la tabla Reservas, el 5% se especificó como 32.500 registros, el 20% como 130.000 registros y el 100% como 650.000 registros.

Al momento de poblar las tablas con el 5% de los datos directamente en SQL Developer se obtuvo un tiempo total de 1612,487 segundos y para el 20% de 6772,4454 segundos. Estos resultados tienen sentido, ya que existe una proporción cercana a 4 entre ambos tiempos de carga.



También se observa un cambio significativo al momento de probar los Requerimientos Funcionales de Consulta. Por ejemplo, probando el RFC 13 (Consultar los buenos clientes), para el 5% de los datos se obtuvo un tiempo de respuesta de 3,94 segundos y, para el 20% de los datos, un tiempo de respuesta de 16,418 segundos.