Universidad de los andes
Sistemas transaccionales
Entrega 2 HotelAndes

FNC 1

Creación de índices

- la selectividad es alta, ya que se seleccionan pocos registros en comparación a la cantidad de registros que se hacen en el join, ya que se seleccionan únicamente los registros de una habitación en un año y en el join salen de todas las habitaciones del hotel.
- Índice por habitación (B+ y primario)

	TABLE_OWNER	INDEX_NAME	POS	COLUMN_NAME
1	ISIS2304D23202320	HABITACIONES_PK	1	ID

- Seguramente fueron creados porque la llave primaria de habitaciones aparece como llave foránea en varias tablas, esto ayudaría en gran medida a operaciones como join

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				2
SORT		AGGREGATE	1	
NESTED LOOPS			1	2
NESTED LOOPS			1	2
TABLE ACCESS	CONSUMOS	FULL	1	2
Filter Predicates				
TO_CHAR(INTERNAL_FUNCTION(CONSUMOS.FECHA),'YYYY')='2003'				
INDEX	HABITACIONES_PK	UNIQUE SCAN	1	0
Access Predicates				
HABITACIONES.ID=CONSUMOS.HABITACIONES_ID				
TABLE ACCESS	HABITACIONES	BY INDEX ROWID	1	0
Filter Predicates				
HABITACIONES.NUMERO=201				
Other XML				

FNC 2

Creación de índices

- La selectividad antes del group by es alta, ya que solo se seleccionan las reservas de cierto periodo de tiempo
- Índice hash por reservas, ya que la inserción es muy recurrente

	TABLE_OWNER	INDEX_NAME	POS	COLUMN_NAME
1	ISIS2304D23202320	RESERVASSERVICIO_PK	1	ID

- Al igual que el requerimiento pasado las+eservas aparecen varias veces como lleva foránea en las habitaciones

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				4
SORT		ORDER BY	1	4
HASH		GROUP BY	1	4
NESTED LOOPS			1	2
NESTED LOOPS			1	2
TABLE ACCESS	RESERVASSERVICIO	FULL	1	2
Filter Predicates				
TO_CHAR(INTERNAL_FUNCTION(RESERVASSERVICIO.FECHA),'YYYY')='2003'				
INDEX	SERVICIOS_PK	UNIQUE SCAN	1	0
Access Predicates				
RESERVASSERVICIO.SERVICIOS_ID=SERVICIOS.ID				
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	1	0
Other XML				
{info}				

FNC 3

Creación de índices

- La selectividad después del group by es alta ya que se seleccionan los registros que sean del ultimo año, además se agruparan muchos registros por habitación. Por tal razón la creación de un índice aquí es crucial
- Índice primario por id de habitación en reservas e índice secundario por fechas, de este modo se agilizaran las dos consultas principales que son el agrupación y comparación de las fechas

	TABLE_OWNER	INDEX_NAME	POS	COLUMN_NAME
1	ISIS2304D23202320	RESERVASALOJAMIENTO_PK	1	ID

-
- NO favorece al requerimiento en el sentido en que no se utilizan los ids de las reservas sino las habitaciones
- Sin indice

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				2
NESTED LOOPS			1	2
NESTED LOOPS			1	2
TABLE ACCESS	RESERVASALOJAMIENTO	FULL	1	2
Filter Predicates				
TO_CHAR(INTERNAL_FUNCTION(RESERVASALOJAMIENTO.FECHAIN),'YYYY')='2003'				
INDEX	HABITACIONES_PK	UNIQUE SCAN	1	0
Access Predicates				
HABITACIONES.ID=RESERVASALOJAMIENTO.HABITACIONES_ID				
TABLE ACCESS	HABITACIONES	BY INDEX ROWID	1	0
Other XML				

-

con indice

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				2
NESTED LOOPS			1	2
NESTED LOOPS			1	2
TABLE ACCESS	RESERVASALOJAMIENTO	FULL	1	2
Filter Predicates				
TO_CHAR(INTERNAL_FUNCTION(RESERVASALOJAMIENTO.FECHAIN),'YYYY')='2003'				
INDEX	HABITACIONES_PK	UNIQUE SCAN	1	0
Access Predicates				
HABITACIONES.ID=RESERVASALOJAMIENTO.HABITACIONES_ID				
TABLE ACCESS	HABITACIONES	BY INDEX ROWID	1	0
Other XML				
{info}				
info type="has_user_tab"				

-

FNC4

Creación de índices

- Aunque la selectividad es alta en cualquiera de los requemientos, el mantenimiento y el espacio que ocupa crear un índice por cada uno de las características seria inviable, por lo tanto seria adecuado crear un índice únicamente por la llave primaria de los servicios como indica Oracle
- Índice primario por pk de servicios y árbol b+ ya que se trabjaran con rangos

	TABLE_OWNER	INDEX_NAME	POS	COLUMN_NAME
1	ISIS2304D23202320	RESERVASSERVICIO_PK	1	ID

FNC5

Creación de índices

- La selectividad de este requerimiento es alta, ya que se busca unicamiente el consumo de un usuario en un rango de fechas entre todos los usuarios. Pero la selectividad es baja entre el mismo usuario.
- Índice primario por usuario b + , índice secundario por fechas b+ ya que se solicitan un rango de fechas

	TABLE_OWNER	INDEX_NAME	POS	COLUMN_NAME
1	ISIS2304D23202320	USUARIOS_PK	1	ID

	TABLE_OWNER	INDEX_NAME	POS	COLUMN_NAME
1	ISIS2304D23202320	RESERVASSERVICIO_PK	1	ID

	TABLE_OWNER	INDEX_NAME	POS	COLUMN_NAME
1	ISIS2304D23202320	HABITACIONES_PK	1	ID

	TABLE_OWNER	INDEX_NAME	POS	COLUMN_NAME
1	ISIS2304D23202320	CUENTACONSUMOS_PK	1	RESERVAALOJAMIENTO_ID
2	ISIS2304D23202320	CUENTACONSUMOS_PK	2	SERVICIO_ID

- Los índices de Oracle son adecuados para realizar los joins necesarios para las consunsultas ya que se aplican a las llaves primarias.

-

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				6
SORT		AGGREGATE	1	
HASH JOIN			1	6
Access Predicates				
CONSUMOS.HABITACIONES_ID=RESERVASALOJAMIENTO.HABITACIONES_ID				
MERGE JOIN		CARTESIAN	1	4
TABLE ACCESS	RESERVASALOJAMIENTO	FULL	1	2
Filter Predicates				
RESERVASALOJAMIENTO.USUARIOS_ID=1				
BUFFER		SORT	1	2
TABLE ACCESS	CONSUMOS	FULL	1	2
TABLE ACCESS	RESERVASALOJAMIENTO	FULL	1	2
Other XML				
{info}				

FNC6

Creación de índices

- La selectividad es alta ya que el conjunto de registros es poca a comparación de todas las reservas
- Índice b+ por fechas ya que se trabajaran por rangos y por habitacion
-

	TABLE_OWNER	INDEX_NAME	POS	COLUMN_NAME
1	ISIS2304D23202320	RESERVASALOJAMIENTO_PK	1	ID

RF7 - ENCONTRAR LOS BUENOS CLIENTES

A la hora de hacer una consulta para encontrar buenos clientes, que son aquellos que han estado en el hotel por lo menos dos semanas (no necesariamente en una sola estadía) o si han consumido más de $15’000.000, durante el último año. Se considera que si es necesario crear un índice en la tabla Reservas Alojamiento este índice sería útil para acelerar la subconsulta que busca los usuarios que han estado en el hotel durante más de dos semanas. El tipo de índice utilizado aquí es un índice B-Tree, que es el tipo de índice predeterminado en Oracle para los datos alfanuméricos y numéricos. Los índices B-Tree son útiles para mejorar el rendimiento de las consultas que utilizan operadores de comparación como =, <>, <, >, <=, y >= .

RF7 utilizando el índice idx_reservasalojamientos_fechain:

```
-- RFC7

SELECT u.id, u.nombre, u.email
FROM Usuarios u
WHERE u.id IN (
    SELECT r.usuarios_id
    FROM ReservasAlojamiento r
    WHERE r.fechain BETWEEN ADD_MONTHS(SYSDATE, -12) AND SYSDATE
    GROUP BY r.usuarios_id
    HAVING COUNT(DISTINCT r.fechain) >= 10
) OR u.id IN (
    SELECT c.id
    FROM Consumos c
    WHERE c.fecha BETWEEN ADD_MONTHS(SYSDATE, -12) AND SYSDATE
    GROUP BY c.id
    HAVING SUM(c.total) > 15000000
);

CREATE INDEX idx_reservasalojamiento_fechain ON ReservasAlojamiento (fechain);
```

Script Output × | Query Result × | Explain Plan ×

SQL | 0.142 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			2	3
NESTED LOOPS			2	3
NESTED LOOPS			2	3
VIEW	SYS.VW_NSO_1		2	3
HASH		UNIQUE	2	3
UNION-ALL				
HASH		GROUP BY	1	1
Filter Predicates				
SUM(C.TOTAL)>15000000				
FILTER				
Filter Predicates				
SYSDATE@!>=ADD_MONTHS(SYSDATE@!,(-12))				
TABLE ACCESS	CONSUMOS	BY INDEX ROWID BATCHED	1	1

RF7 sin utilizar el índice:

```
-- RFC7

SELECT u.id, u.nombre, u.email
FROM Usuarios u
WHERE u.id IN (
    SELECT r.usuarios_id
    FROM ReservasAlojamiento r
    WHERE r.fechain BETWEEN ADD_MONTHS(SYSDATE, -12) AND SYSDATE
    GROUP BY r.usuarios_id
    HAVING COUNT(DISTINCT r.fechain) >= 10
) OR u.id IN (
    SELECT c.id
    FROM Consumos c
    WHERE c.fecha BETWEEN ADD_MONTHS(SYSDATE, -12) AND SYSDATE
    GROUP BY c.id
    HAVING SUM(c.total) > 15000000
);

CREATE INDEX idx_reservasalojamiento_fechain ON ReservasAlojamiento (fechain);
Drop index idx_reservasalojamiento_fechain;
```

Script Output × | Query Result × | Explain Plan ×

SQL | 0.103 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			2	5
NESTED LOOPS			2	5
NESTED LOOPS			2	5
VIEW	SYS.VW_NSO_1		2	5
HASH		UNIQUE	2	5
UNION-ALL				
HASH		GROUP BY	1	1
Filter Predicates				
SUM(C.TOTAL)>15000000				
FILTER				
Filter Predicates				
SYSDATE@!>=ADD_MONTHS(SYSDATE@!,(-12))				
TABLE ACCESS	CONSUMOS	BY INDEX ROWID BATCHED	1	1

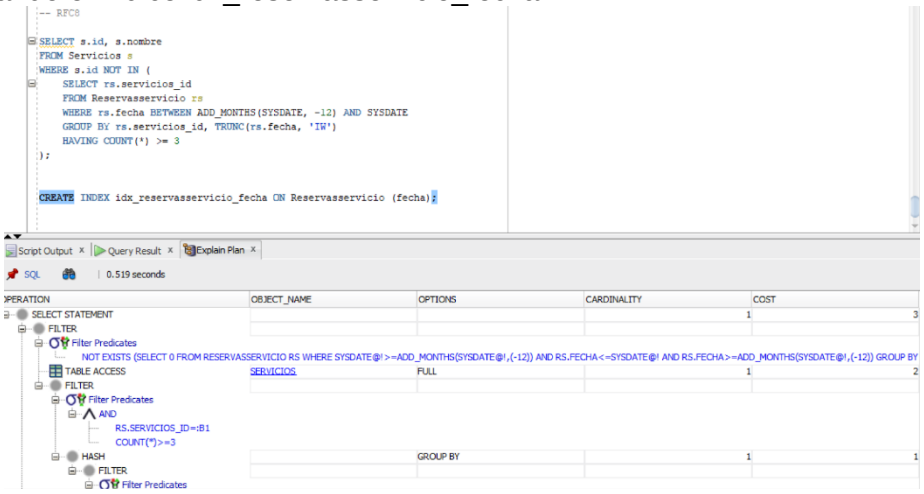
RF8 - ENCONTRAR LOS SERVICIOS QUE NO TIENEN MUCHA DEMANDA

Para la implementación del requerimiento se decidió hacerlo mediante una consulta que seleccione todos los servicios que hayan sido solicitados menos de 3 veces semanales durante el último año de operación de HotelAndes. Primero, se necesita contar el número de veces que cada `servicios_id` aparece en la tabla `Reservasservicio` para cada semana del último año. Luego, se seleccionan aquellos servicios que aparecen menos de 3 veces por semana. En esta consulta, `TRUNC(rs.fecha, 'IW')` se utiliza para obtener el inicio de la semana para cada fecha en la columna `fecha` de la tabla `Reservasservicio`. Por otro lado, `ADD_MONTHS(SYSDATE, -12)` se utiliza para obtener la fecha exacta de un año antes de la fecha actual, y SYSDATE se utiliza para obtener la fecha y hora actuales.

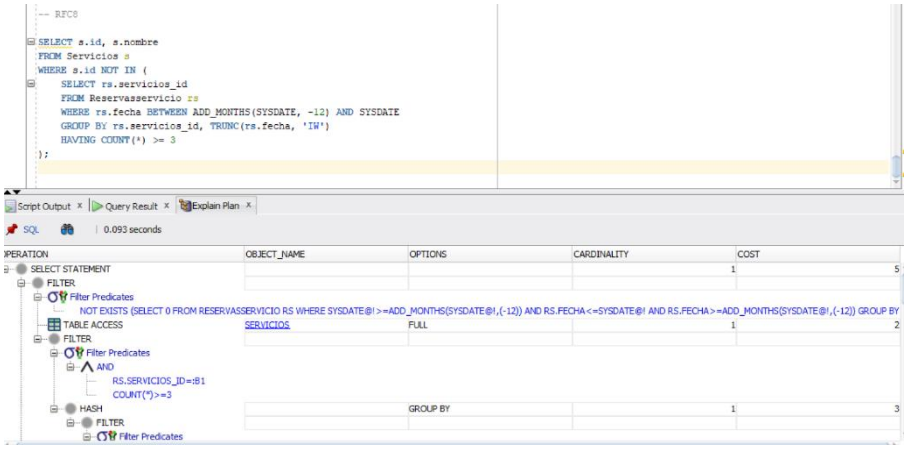
En este requerimiento El índice `idx_reservasservicio_fecha` en la tabla `Reservasservicio` se crea para mejorar el rendimiento de las consultas que filtran por la columna `fecha`. En el contexto, este índice sería útil para acelerar la subconsulta que

busca los servicios que han sido solicitados durante el último año. El tipo de índice utilizado aquí es un índice B-Tree.

RF8 utilizando el índice idx_reservasservicio_fecha:



RF8 sin indice



RF9 - CONSULTAR CONSUMO EN HOTELANDES

Para el desarrollo de este requerimiento se hizo un consulta en SQL la cual selecciona el ID y el nombre del usuario, el nombre del servicio y el número de veces que el usuario consumió ese servicio. Luego, une las tablas `usuarios`, `reservasalojamiento`, `habitaciones`, `reservasservicio`, `servicios` y `consumos` en función de las relaciones entre ellas. Después, filtra los consumos por la fecha y sólo considera los consumos que están dentro del rango de fechas proporcionado por el usuario. Posteriormente, agrupa los resultados por usuario y servicio. Finalmente, ordena los resultados según el criterio proporcionado por el usuario, y por el nombre del usuario en orden ascendente. Para optimizar esta consulta, se puede tener en cuenta la creación de índices en las columnas que se utilizan en las cláusulas JOIN y WHERE, estos pueden acelerar las operaciones de lectura en la base de datos al proporcionar una forma eficiente de localizar filas en una tabla. En este caso, se consideró la creación de índices en las columnas `fecha` de la

tabla `consumos`, `idusuario` y `habitaciones_id` de la tabla `reservasalojamiento`, y `habitaciones_id` de la tabla `reservasservicio`

RF9 sin indice

Script Output × | Query Result × | Explain Plan ×

SQL | 0.585 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	7
SORT		GROUP BY	1	7
FILTER				
Filter Predicates				
TO_DATE(:FECHA_FIN,'DD-MM-YYYY')>=TO_DATE(:FECHA_INICIO,'DD-MM-YYYY')				
NESTED LOOPS			1	6
NESTED LOOPS			1	6
HASH JOIN			1	6
Access Predicates				
AND				
C.HABITACIONES_ID=RS.HABITACIONES_ID				
RA.HABITACIONES_ID=C.HABITACIONES_ID				
MERGE JOIN		CARTESIAN	1	4
NESTED LOOPS			1	2
NESTED LOOPS			1	2
TABLE ACCESS	RESERVASSERVICIO	FULL	1	2
INDEX	SERVICIOS_PK	UNIQUE SCAN	1	0
Access Predicates				
RS.SERVICIOS_ID=S.ID				
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	1	0
BUFFER		SORT	1	4
TABLE ACCESS	RESERVASALOJAMIENTO	FULL	1	2
TABLE ACCESS	CONSUMOS	FULL	1	2
Filter Predicates				
AND				
C.FECHA>=TO_DATE(:FECHA_INICIO,'DD-MM-YYYY')				
C.FECHA<=TO_DATE(:FECHA_FIN,'DD-MM-YYYY')				
INDEX	USUARIOS_PK	UNIQUE SCAN	1	0
Access Predicates				
U.ID=RA.USUARIOS_ID				
TABLE ACCESS	USUARIOS	BY INDEX ROWID	1	0
Other XML				
{info}				

RF9 con indice

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	4
SORT		GROUP BY	1	4
FILTER				
Filter Predicates				
TO_DATE(:FECHA_FIN,'DD-MM-YYYY')>=TO_DATE(:FECHA_INICIO,'DD-MM-YYYY')				
NESTED LOOPS			1	3
NESTED LOOPS			1	3
NESTED LOOPS			1	3
NESTED LOOPS			1	3
NESTED LOOPS			1	3
TABLE ACCESS	RESERVASSERVICIO	FULL	1	2
TABLE ACCESS	CONSUMOS	BY INDEX ROWID BATCHED	1	1
Filter Predicates				
C.HABITACIONES_ID=RS.HABITACIONES_ID				
INDEX	IDX_CONSUMOS_FECHA	RANGE SCAN	1	1
Access Predicates				
AND				
C.FECHA>=TO_DATE(:FECHA_INICIO,'DD-MM-YYYY')				
C.FECHA<=TO_DATE(:FECHA_FIN,'DD-MM-YYYY')				
TABLE ACCESS	RESERVASALOJAMIENTO	BY INDEX ROWID BATCHED	1	0
INDEX	IDX_RESERVASALOJAMIENTO_HABIT...	RANGE SCAN	1	0
Access Predicates				
RA.HABITACIONES_ID=C.HABITACIONES_ID				
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	1	0
INDEX	SERVICIOS_PK	UNIQUE SCAN	1	0
Access Predicates				
RS.SERVICIOS_ID=S.ID				
INDEX	USUARIOS_PK	UNIQUE SCAN	1	0
Access Predicates				
U.ID=RA.USUARIOS_ID				
TABLE ACCESS	USUARIOS	BY INDEX ROWID	1	0
Other XML				
{info}				

RF10 - CONSULTAR CONSUMO EN HOTELANDES – RFC9-V2

En esta consulta, se reemplaza `:fecha_inicio`, `:fecha_fin`, `:nombre_servicio` y `:criterio_ordenamiento` con los valores proporcionados por el usuario. Esta consulta devuelve el ID y el nombre del usuario que no consumió un servicio específico en el rango de fechas especificado. La consulta se realiza uniendo las tablas `usuarios`, `reservasalojamiento`, `habitaciones`, `reservasservicio`, `servicios` y `consumos` en función de las relaciones entre ellas. Luego, se filtran los consumos por la fecha y el nombre del servicio, y se excluyen los usuarios que consumieron ese servicio. Finalmente, se ordenan los resultados según el criterio proporcionado por el usuario.

RF10 sin indice

Script Output × | Query Result × | Explain Plan ×

SQL | 0.069 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			12	8
MERGE JOIN		ANTI	12	8
TABLE ACCESS	USUARIOS	BY INDEX ROWID	12	1
INDEX	USUARIOS_PK	FULL SCAN	12	1
SORT		UNIQUE	1	7
Access Predicates				
U.ID=ITEM_1				
Filter Predicates				
U.ID=ITEM_1				
VIEW	SYS.VW_SQ_1		1	6
FILTER				
Filter Predicates				
TO_DATE(:FECHA_FIN,'DD-MM-YYYY')>=TO_DATE(:FECHA_INICIO,'DD-MM-YYYY')				
HASH JOIN			1	6
Access Predicates				
AND				
RA.HABITACIONES_ID=C.HABITACIONES_ID				
RA.HABITACIONES_ID=RS.HABITACIONES_ID				
MERGE JOIN		CARTESIAN	1	4
NESTED LOOPS		SEMI	1	2
TABLE ACCESS	RESERVASSERVICIO	FULL	1	2
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	1	0
Filter Predicates				
S.NOMBRE=:NOMBRE_SERVICIO				
INDEX	SERVICIOS_PK	UNIQUE SCAN	1	0
Access Predicates				
RS.SERVICIOS_ID=S.ID				
BUFFER		SORT	1	4
TABLE ACCESS	CONSUMOS	FULL	1	2
Filter Predicates				
AND				
C.FECHA>=TO_DATE(:FECHA_INICIO,'DD-MM-YYYY')				
C.FECHA<=TO_DATE(:FECHA_FIN,'DD-MM-YYYY')				

RF10 con indices

```
SELECT
    u.id AS usuario_id,
    u.nombre AS nombre_usuario
FROM
    usuarios u
WHERE
    NOT EXISTS (
        SELECT 1
        FROM
            reservasalojamiento ra
        JOIN
            habitaciones h ON ra.habitaciones_id = h.id
        JOIN
            reservasservicio rs ON h.id = rs.habitaciones_id
        JOIN
            servicios s ON rs.servicios_id = s.id
        JOIN
            consumos c ON h.id = c.habitaciones_id
```

Query Result × | Explain Plan ×

SQL | 0.055 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			12	3
MERGE JOIN		ANTI	12	3
TABLE ACCESS	USUARIOS	BY INDEX ROWID	12	1
INDEX	USUARIOS_PK	FULL SCAN	12	1
SORT		UNIQUE	1	2
Access Predicates				
U.ID=ITEM_1				
Filter Predicates				
U.ID=ITEM_1				
VIEW	SYS.VW_SQ_1		1	1
FILTER				
Filter Predicates				
TO_DATE(:FECHA_FIN,'DD-MM-YYYY')>=TO_DATE(:FECHA_INICIO,'DD-MM-YYYY')				
NESTED LOOPS			1	1
NESTED LOOPS			1	1
NESTED LOOPS			1	1

El índice idx_consumos_fecha, es un índice no agrupado que está en la columna `fecha` de la tabla `consumos`. Cuando se realiza una consulta que involucra esta columna en las cláusulas JOIN o WHERE, el motor de la base de datos puede buscar rápidamente los datos correspondientes utilizando el índice. Esto hace que la consulta se ejecute más rápido y consuma menos recursos del sistema

En cuanto a los indices creados por Oracle, podemos observar los índices Anti y Semi, estos son tipos de operaciones de unión que se utilizan en las consultas SQL.

- Un Anti-Join es una operación de unión que devuelve las filas de la primera tabla que no tienen coincidencias en la segunda tabla.
- Un Semi-Join es una operación de unión que devuelve las filas de la primera tabla que tienen al menos una coincidencia en la segunda tabla.

En el contexto de los índices, los Anti-Joins y Semi-Joins pueden beneficiarse de los índices no agrupados. Los índices no agrupados pueden proporcionar un rendimiento más

rápido para estas operaciones de unión porque permiten que el motor de la base de datos busque rápidamente las filas correspondientes utilizando el índice, en lugar de tener que escanear toda la tabla.

RF11 - CONSULTAR FUNCIONAMIENTO

Para este requerimiento se decidió separar por consultas la información que se solicita es decir, cada una de las consultas están diseñadas para determinar el servicio más y menos consumido y la habitación más y menos solicitada en una base de datos de hotel. La primera consulta selecciona la fecha y el nombre del servicio, y cuenta el número de veces que se ha consumido cada servicio, ordenando los resultados en orden descendente. La segunda consulta hace lo mismo, pero ordena los resultados en orden ascendente. La tercera consulta selecciona la fecha de inicio de la reserva y el número de la habitación, y cuenta el número de veces que se ha solicitado cada habitación, ordenando los resultados en orden descendente. La cuarta consulta hace lo mismo, pero ordena los resultados en orden ascendente. Cada consulta devuelve solo la primera fila de los resultados ordenados.

```
-- Servicio mas consumido
SELECT
    c.fecha,
    s.nombre,
    COUNT(*) AS num_consumos
FROM
    consumos c
JOIN
    reservasservicio rs ON c.habitaciones_id = rs.habitaciones_id
JOIN
    servicios s ON rs.servicios_id = s.id
GROUP BY
    c.fecha, s.nombre
ORDER BY
    num_consumos DESC
FETCH FIRST 1 ROWS ONLY;
```

Script Output | Query Result | Explain Plan

SQL | 0.096 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	7
SORT		ORDER BY	1	7
VIEW	SYS.null		1	6
Filter Predicates				
from$_subquery$_006.rowlimit_$$_rownumber<=1				
WINDOW		SORT PUSHED RANK	1	6
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC)<=1				
HASH		GROUP BY	1	6
NESTED LOOPS			1	4
NESTED LOOPS			1	4
HASH JOIN			1	4
Access Predicates				
C.HABITACIONES_ID=RS.HABITACIONES_ID				
TABLE ACCESS	CONSUMOS	FULL	1	2
TABLE ACCESS	RESERVASSERVICIO	FULL	1	2
INDEX	SERVICIOS_PK	UNIQUE SCAN	1	0

```sql
-- Servicio menos consumido
SELECT
    TO_CHAR(c.fecha, 'IW') AS semana,
    s.nombre,
    COUNT(*) AS num_consumos
FROM
    consumos c
JOIN
    reservasservicio rs ON c.habitaciones_id = rs.habitaciones_id
JOIN
    servicios s ON rs.servicios_id = s.id
GROUP BY
    c.fecha, s.nombre
ORDER BY
    num_consumos ASC
FETCH FIRST 1 ROWS ONLY;

-- Habitación más solicitada
```

Script Output | Query Result | Explain Plan

SQL | 0.058 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	7
SORT		ORDER BY	1	7
VIEW	SYS.null		1	6
Filter Predicates				
from$_subquery$_006.rowlimit_$$_rownumber<=1				
WINDOW		SORT PUSHED RANK	1	6
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*))<=1				
HASH		GROUP BY	1	6
NESTED LOOPS			1	4
NESTED LOOPS			1	4
HASH JOIN			1	4
Access Predicates				
C.HABITACIONES_ID=RS.HABITACIONES_ID				

```sql
FETCH FIRST 1 ROWS ONLY;

-- Habitación más solicitada
SELECT
    TO_CHAR(r.fechain, 'IW') AS semana,
    h.numero,
    COUNT(*) AS num_reservas
FROM
    reservasalojamiento r
JOIN
    habitaciones h ON r.habitaciones_id = h.id
GROUP BY
    r.fechain, h.numero
ORDER BY
    num_reservas DESC
FETCH FIRST 1 ROWS ONLY;

-- Habitación menos solicitada
SELECT
```

Script Output | Query Result | Explain Plan

SQL | 0.039 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	5
SORT		ORDER BY	1	5
VIEW	SYS.null		1	4
Filter Predicates				
from$_subquery$_004.rowlimit_$$_rownumber<=1				
WINDOW		SORT PUSHED RANK	1	4
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC)<=1				
HASH		GROUP BY	1	4
NESTED LOOPS			1	2
NESTED LOOPS			1	2
TABLE ACCESS	RESERVASALOJAMIENTO	FULL	1	2
INDEX	HABITACIONES_PK	UNIQUE SCAN	1	0
Access Predicates				

```sql
-- Habitación menos solicitada
SELECT
    TO_CHAR(r.fechain, 'IW') AS semana,
    h.numero,
    COUNT(*) AS num_reservas
FROM
    reservasalojamiento r
JOIN
    habitaciones h ON r.habitaciones_id = h.id
GROUP BY
    r.fechain, h.numero
ORDER BY
    num_reservas ASC
FETCH FIRST 1 ROWS ONLY;
```

Script Output | Query Result | Explain Plan

SQL | 0.069 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	5
SORT		ORDER BY	1	5
VIEW	SYS.null		1	4
Filter Predicates				
from$_subquery$_004.rowlimit_$$_rownumber<=1				
WINDOW		SORT PUSHED RANK	1	4
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*))<=1				
HASH		GROUP BY	1	4
NESTED LOOPS			1	2
NESTED LOOPS			1	2
TABLE ACCESS	RESERVASALOJAMIENTO	FULL	1	2
INDEX	HABITACIONES_PK	UNIQUE SCAN	1	0
Access Predicates				

Documentación de los datos

Se realizo un script que automáticamente hace datos que obedecen fielmente al modelo relacional y a la estructura de la DB, se crearon: 100.000 usuarios, 5 pisos, 50 habitaciones por piso, 300.000 reservas de alojamiento, 200.000 reservas de servicios y 100.000 consumos. Para un total de 600.250 registros en la DB, Cabe resaltar que los datos obedecen las reglas del negocio. Para el proceso de importación de los datos, se crearon archivos SQL (con las operaciones para cada inserción) y csv (con cada uno de los registros), pero por problemas técnicos se termino optando por la inserción con las sentencias SQL.