

Grupo B3
Manuela Lizcano
Juan Diego Lozano
Mariana Pineda

ENTREGA 2 SISTEMAS TRANSACCIONALES

Requerimiento 1

Índices creados:

Índice	Tipo	Explicación
HABITACIONES. nHabitación	Primario y simple	Este índice se crea teniendo en cuenta que se quiere acceder a la información de una habitación es específico, por lo que al realizar los JOINS con las demás tablas se tendrá la información de los consumos de cada habitación agrupados y será mucho más fácil acceder a ellos. De igual manera, este índice es simple ya que se crea en una sola columna de la tabla y en la llave primaria de esta por lo que es in índice primario. Este índice fue creado por SQL Developer.
CONSUMOS. (nHabitacion, idservicio)	Secundario y compuesto	Este índice se crearía para mejorar la eficiencia del JOIN entre la tabla de CONSUMOS y la tabla de HABITACIONES. Agruparía todas las habitaciones en un mismo índice por lo que acceder a los consumos de cada una de las habitaciones seria más eficiente. Además, agrupa los servicios consumidos por cada una de las habitaciones y los servicios que consumió así poder acceder a cada servicio y su costo de mejor manera. De igual manera este índice es secundario ya que no hace referencia directamente a una llave primaria de la tabla y este es compuesto debido a que hace referencia a dos columnas de la tabla.

Sentencia SQL:

```
SELECT h.nHabitacion, sum(s.costoservicio) as tipohabitacion
FROM habitaciones h
INNER JOIN consumos c ON(h.nHabitacion = c.nHabitacion)
RIGHT JOIN servicios s ON(c.idservicio = s.idservicio)
WHERE h.nHabitacion = 450
GROUP BY h.nHabitacion
ORDER BY h.nHabitacion;
```

Distribución de datos:

En este caso se quieren buscar un cuanto han generado cada una de las habitaciones del hotel por medio de servicios. El costo de cada uno de los servicios se encuentra en tabla SERVICIOS y el consumo hecho por cada una de las habitaciones se encuentra en la tabla CONSUMOS. Con lo anterior se hace un JOIN por el campo de idservicio. Luego las habitaciones del hotel se encuentran en la tabla HABITACIONES por lo que se hace un JOIN en el cual se tienen los consumos cada una de las habitaciones. Finalmente, para calcular el dinero que genera cada una de las habitaciones se suman los costos en los que ha incurrido cada una de las habitaciones en la operación del hotel.

Plan de ejecución Oracle:

SELECT STATEMENT			250	74
SORT		GROUP BY NOSORT	250	74
MERGE JOIN			250	74
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	100	2
INDEX	SERVICIOS_PK	FULL SCAN	100	1
SORT		JOIN	250	72
Access Predicates				
C.IDSERVICIO=S.IDSERVICIO				
Filter Predicates				
C.IDSERVICIO=S.IDSERVICIO				
TABLE ACCESS	CONSUMOS	FULL	250	71
Filter Predicates				
C.NHABITACION=106				

Con respecto a este plan de ejecución se hace uso de un MERGE JOIN esto ya que los datos están ordenados por la columna en la que esta su llave primaria. De igual manera, se hace uso del índice SERVICIOS_PK el cual ayuda a acceder de manera más eficiente a el costo de cada uno de los servicios correspondientes a los consumos hechos por cada una de las habitaciones.

Tiempos:

Prueba 1:

nHabitacion = 106

Tiempo = 0.045 segs

Prueba 2:

nHabitacion = 300

Tiempo = 0.012 segs

Prueba 3:

nHabitacion = 450

Tiempo = 0.013 segs

Mejora con índice creado:

En este caso se tiene el índice llamado CONSUMOHABITACIONES el cual es un índice compuesto por las columnas de la tabla de CONSUMOS entre las columnas de id servicio y nHabitacion. Este índice ayuda a que esta consulta se optimice reduciendo su costo a tan solo 5 lo que supone una reducción del costo de esta consulta del 94,24%. De igual manera, la tabla también

hace uso del índice de SERVICIOS_PK el cual ya fue previamente creado por la consulta para poder acceder al costo de los servicios de manera más fácil.

SELECT STATEMENT			250	5
SORT		GROUP BY NOSORT	250	5
MERGE JOIN			250	5
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	100	2
INDEX	SERVICIOS_PK	FULL SCAN	100	1
SORT		JOIN	250	3
Access Predicates	C.IDSERVICIO=S.IDSERVICIO			
Filter Predicates	C.IDSERVICIO=S.IDSERVICIO			
INDEX	CONSUMAHABITACIONES	RANGE SCAN	250	2
Access Predicates	C.NHABITACION=106			

Requerimiento 2

Índices creados:

Índice	Tipo	Explicación
CONSUMOS (idconsumo, idservicio)	Secundario y compuesto	En este caso se tiene un índice compuesto el cual hace uso de las columnas de idconsumo y id servicio. Este índice sirve para agrupar cada uno de los consumos correspondientes a un servicio en específico. Además, ayuda a hacer más eficientes los JOIN que se hacen a la tabla de servicio y a realizar la función de conteo de las veces que se consume un servicio para poder ver los 20 servicios más populares.

Sentencia SQL:

```
SELECT s.nombre, s.dtype, count(c.idconsumo)
FROM servicios s
RIGHT JOIN consumos c ON(s.idservicio = c.idservicio)
GROUP BY s.nombre, s.dtype
ORDER BY count(c.idconsumo) DESC
FETCH FIRST 20 ROWS ONLY;
```

Distribución de datos:

En este caso la distribución de los datos que se van a utilizar esta en las tablas se SERVICIOS y CONSUMOS en donde se hace un JOIN por medio del campo idservicio y se toman los servicios los cuales tengan la mayor cantidad de consumos.

Plan de ejecución Oracle:

SELECT STATEMENT			20	83
SORT	ORDER BY		20	83
VIEW	SYS.null		20	82
Filter Predicates	from\$_subquery\$_004.rowlimit_\$_rownumber<=20			
WINDOW				
Filter Predicates	ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC)<=20			
HASH				
HASH JOIN				
Access Predicates	S.IDSERVICIO(+)=C.IDSERVICIO			
TABLE ACCESS	SERVICIOS	FULL	100	4
TABLE ACCESS	CONSUMOS	FULL	100000	71

En este caso, la consulta se hace por medio de un HASH JOIN en el que se unen las tablas de servicio y consumos, en este caso no se hace uso de ningún índice pero se tiene un costo bastante elevado.

Tiempos:

Prueba 1:

Tiempo: 0.059 segundos

Solo se tiene una prueba debido a que no existen parámetros de entrada los cuales puedan variar.

Mejora con índice creado:

SELECT STATEMENT			20	64
SORT	ORDER BY		20	64
VIEW	SYS.null		20	63
Filter Predicates	from\$_subquery\$_004.rowlimit_\$_rownumber<=20			
WINDOW				
Filter Predicates	ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC)<=20			
HASH				
HASH JOIN				
Access Predicates	S.IDSERVICIO(+)=C.IDSERVICIO			
TABLE ACCESS	SERVICIOS	FULL	100	4
INDEX	CONSUMOSERVICIOS	FAST FULL SCAN	100000	51

El índice compuesto creado se usa para poder acceder de manera conjunta sobre cada uno de los servicios en la tabla de consumos para así poder contar cada vez que aparece cada servicio de manera más eficiente. Con esto el costo de este requerimiento se redujo en un 22,89%.

Requerimiento 3

Índices creados

Índice	Tipo	Explicación
RESERVAS.(NHABITACION)	Simple y secundario	Este índice se crea considerando que esta consulta requiere de la información relacionada con cada una de las habitaciones en el hotel, y esto

		facilitaría las búsquedas de la consulta. Este índice es simple y secundario considerando que se realiza únicamente para una columna de las relaciones y es secundario debido a que no se realiza para la primary key de la relación de reservas.
--	--	---

Sentencia SQL

```
SELECT NHABITACION, COUNT(*)/(SELECT COUNT(*)
FROM RESERVAS) AS CANT_RESERVAS_HAB
FROM HABITACIONES
NATURAL JOIN RESERVAS
GROUP BY NHABITACION;
```

Distribución de datos:

Este requerimiento busca hallar el índice de ocupación para cada una de las habitaciones del hotel. Con el objetivo de lograr dicha consulta, en primer lugar, se tiene que tener en cuenta que cada habitación durante un año cuenta con múltiples reservas. A partir de esto, también tenemos que tener en cuenta que cada reserva tiene asociada una habitación por esta razón se realizó un NATURAL JOIN entre las dos relaciones para poder determinar la cantidad de veces en la que cada una de las habitaciones estuvo ocupada, por esta razón realizamos un GROUP BY para cada una de las habitaciones. Para poder mostrar la fracción de ocupación de cada habitación, se realizó un SUBCONSULTA con el objetivo de hallar la cantidad total de reservas y con este valor hallar el índice de ocupación de cada una de las habitaciones.

Plan de ejecución Oracle:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				400 116
SORT		AGGREGATE		1
INDEX	RESERVAS_PK	FAST FULL SCAN		50000 19
HASH		GROUP BY		400 116
TABLE ACCESS	RESERVAS	FULL		50000 95

Tiempos: 0.055 segundos

Mejora con índice creado:

A partir de la creación del índice, se puede ver una mejora en el costo de 93, considerando que el índice permite realizar las búsquedas de una forma mucho más rápida al tener indexados los valores asociados con los números de las habitaciones. Después de realizar el índice, el tiempo que tomo en realizar la consulta fue de 0,017 segundos.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				400 43
SORT		AGGREGATE		1
INDEX	RESERVAS_PK	FAST FULL SCAN		50000 19
HASH		GROUP BY		400 43
INDEX	RESERVAS_INDICE	FAST FULL SCAN		50000 21

Requerimiento 4

Índices creados

Índice	Tipo	Explicación
CONSUMOS(idservicio, nHabitacion)	Secundario y compuesto.	En este caso se tiene un índice compuesto en la tabla de consumos entre las columnas de idservicio y nHabitación, con esto se facilita el acceso a cada uno de los servicios consumidos por las habitaciones, por lo que la búsqueda de la habitación y de los servicios que son consumidos por estas habitaciones es más eficiente. Este es un índice secundario debido a que no está exclusivamente en la llave primaria de la tabla, además es compuesto debido a que no se encuentra solo en una de las columnas de la tabla.

Sentencia SQL:

```
SELECT s.nombre, s.dtype, s.costoservicio
FROM servicios s
JOIN consumos c ON(s.idservicio = c.idservicio)
JOIN reservasservicio r ON(r.idservicio = s.idservicio)
WHERE (s.costoservicio < 100000
AND s.costoservicio > 60000)
AND
(c.nHabitacion = 105)
AND
(r.fecha > '04/10/2024' AND r.fecha < '10/12/2024'
)
GROUP BY s.nombre, s.dtype, s.costoservicio;
```

Distribución de datos:

Para este requerimiento los datos están distribuidos en tres tablas, SERVICIOS, CONSUMOS y RESERVASSERVICIOS. En este caso como se quiere ver los servicios con un rango de costos específicos o que se hayan consumido o reservado por una habitación en específico entre unas fechas específicas. Para esta consulta se realizan dos operaciones de JOIN, una de la tabla servicios con la tabla consumos por la columna de ambas tablas que contiene el id servicio y un segundo JOIN en el que se une la tabla de reserva servicio con los servicios correspondientes.

Plan de ejecución Oracle:

SELECT STATEMENT			55	170
HASH		GROUP BY	55	170
HASH JOIN		SEMI	55	169
Access Predicates				
R.IDSERVICIO=S.IDSERVICIO				
MERGE JOIN		SEMI	55	74
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	55	2
Filter Predicates				
AND				
S.COSTOSERVICIO>50000				
S.COSTOSERVICIO<90000				
INDEX	SERVICIOS_PK	FULL SCAN	100	1
SORT		UNIQUE	250	72
Access Predicates				
S.IDSERVICIO=C.IDSERVICIO				
Filter Predicates				
S.IDSERVICIO=C.IDSERVICIO				
TABLE ACCESS	CONSUMOS	FULL	250	71
Filter Predicates				
C.NHABITACION=300				
TABLE ACCESS	RESERVASSERVICIO	FULL	646	95
Filter Predicates				
AND				
R.FECHA<'01/08/2024'				
R.FECHA>'01/05/2024'				

En este caso el requerimiento se hace por medio de un merge join por que se tienen los datos de cada tabla ordenados. Así mismo, se hace uso del índice de SERVICIOS PK el cual ayuda a que se pueda acceder a la información de los servicios como el nombre, tipo y costo.

Tiempos:

Prueba 1:

Numero habitación = 300

Costo superior = 90000

Costo inferior = 50000

Fecha superior = '01/08/2024'

Fecha inferior = '01/05/2024'

Tiempo: 0.043 segs

Prueba 2:

Numero habitación = 450

Costo superior = 20000

Costo inferior = 60000

Fecha superior = '09/07/2024'

Fecha inferior = '02/07/2024'

Tiempo: 0.04 segs

Prueba 3:

Numero habitación = 105

Costo superior = 60000

Costo inferior = 100000

Fecha superior = '10/12/2024'

Fecha inferior = '04/10/2024'

Tiempo: 0.045 segs

Mejora con índice creado:

SELECT STATEMENT			55	147
HASH		GROUP BY	55	147
HASH JOIN		SEMI	55	146
Access Predicates				
R.IDSERVICIO=S.IDSERVICIO				
MERGE JOIN		SEMI	55	52
TABLE ACCESS	SERVICIOS	BY INDEX ROWID	55	2
Filter Predicates				
AND				
S.COSTOSERVICIO>50000				
S.COSTOSERVICIO<90000				
INDEX	SERVICIOS_PK	FULL SCAN	100	1
UNIQUE			250	50
Access Predicates				
S.IDSERVICIO=C.IDSERVICIO				
Filter Predicates				
S.IDSERVICIO=C.IDSERVICIO				
INDEX	SERVHABIT	FAST FULL SCAN	250	49
Filter Predicates				
C.NHABITACION=300				
TABLE ACCESS	RESERVASSERVICIO	FULL	646	95
Filter Predicates				
AND				
R.FECHA<'01/08/2024'				
R.FECHA>'01/05/2024'				

En este caso la creación de este índice el cual tiene como nombre SERVHABIT donde se puede acceder con mayor facilidad a cada uno de los servicios que consumió una habitación. Con este índice el costo se pudo reducir en un 13,53% lo que es bueno para una consulta la cual usa muchos recursos.

Requerimiento 5

Índices creados:

Índice	Tipo	Explicación
reservasservicio.(idservicio, fecha)	compuesto	En de reservas servicio con servicio se está realizando un acceso completo y se está teniendo una cardinalidad de 250. En este caso se crea el índice para filtrar de mejor manera por ranog las fechas y que sea más eficiente

Sentencia SQL:

```

SELECT DISTINCT c.idconsumo, rs.fecha, u.idusuario, u.nombre
FROM consumos c
JOIN reservas r ON (r.nhabitacion = c.nHabitacion)
JOIN servicios s ON (s.idservicio = c.idservicio)
JOIN reservasservicio rs ON (rs.idservicio = s.idservicio)
JOIN usuarios u ON (u.idusuario = r.idusuario)
WHERE u.idusuario = :id
      AND TO_DATE (rs.fecha, 'MM/DD/YYYY') BETWEEN TO_DATE(:inicio, 'MM/DD/YYYY')
      AND TO_DATE (:fin, 'MM/DD/YYYY')
ORDER BY rs.fecha;

```

Distribución de datos:

En esta consulta se quiere obtener los consumos que ha tenido un cliente en un rango de fechas específico. Para lograr esto se tiene que revisar la tabla de consumos junto a las tablas de reservas, de servicios y de reservas de servicio. Esto se debe a que un consumo se encuentra en los servicios, por lo tanto, cuando se cruce cada consumo con su respectiva reserva se logra obtener la fecha de este consumo y tener las condiciones que se solicitan.

Plan de ejecución Oracle:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1707	268
SORT		ORDER BY	1707	268
FILTER		UNIQUE	1707	267
Filter Predicates				
TO_DATE(:FIN,'MM/DD/YYYY')>=TO_DATE(:INICIO,'MM/DD/YYYY')				
HASH JOIN			1707	266
Access Predicates				
RS.IDSERVICIO=C.IDSERVICIO				
TABLE ACCESS	RESERVASSERVICIO	FULL	250	98
Filter Predicates				
AND				
TO_DATE(RS.FECHA,'MM/DD/YYYY')>=TO_DATE(:INICIO,'MM/DD/YYYY')				
TO_DATE(RS.FECHA,'MM/DD/YYYY')<=TO_DATE(:FIN,'MM/DD/YYYY')				
HASH JOIN		RIGHT SEMI	683	168
Access Predicates				
R.NHABITACION=C.NHABITACION				
TABLE ACCESS	RESERVAS	FULL	3	95
Filter Predicates				
R.IDUSUARIO=TO_NUMBER(:ID)				
NESTED LOOPS			100000	73
TABLE ACCESS	USUARIOS	BY INDEX ROWID	1	2
INDEX	USUARIOS_PK	UNIQUE SCAN	1	1
Access Predicates				
U.IDUSUARIO=TO_NUMBER(:ID)				
TABLE ACCESS	CONSUMOS	FULL	100000	71

Tiempos:

La consulta se demora 0,059 segundos para los siguientes valores:

Id del usuario = 781

Fecha de inicio = 10/10/2024

Fecha de fin = 12/12/2024

Mejora con índice creado:

En el plan de ejecución con el índice creado se puede ver que el costo disminuyo en 28. Esto se debe a que como se puede ver en el acceso a las tablas mencionadas de reservas servicio y servicios bajo el costo. El tiempo de ejecución termina siendo

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1707	240
SORT		ORDER BY	1707	240
SORT		UNIQUE	1707	239
FILTER				
Filter Predicates				
TO_DATE(:FIN,'MM/DD/YYYY')>=TO_DATE(:INICIO,'MM/DD/YYYY')				
HASH JOIN			1707	238
Access Predicates				
RS.IDSERVICIO=C.IDSERVICIO				
INDEX	IDX_RESERVASSERVICIO_ID...	FAST FULL SCAN	250	70
Filter Predicates				
AND				
TO_DATE(RS.FECHA,'MM/DD/YYYY')>=TO_DATE(:INICIO,'MM/DD/YYYY')				
TO_DATE(RS.FECHA,'MM/DD/YYYY')<=TO_DATE(:FIN,'MM/DD/YYYY')				
HASH JOIN		RIGHT SEMI	683	168
Access Predicates				
R.NHABITACION=C.NHABITACION				
TABLE ACCESS	RESERVAS	FULL	3	95
Filter Predicates				
R.IDUSUARIO=TO_NUMBER(:ID)				
NESTED LOOPS			100000	73
TABLE ACCESS	USUARIOS	BY INDEX ROWID	1	2
INDEX	USUARIOS_PK	UNIQUE SCAN	1	1
Access Predicates				
U.IDUSUARIO=TO_NUMBER(:ID)				
TABLE ACCESS	CONSUMOS	FULL	100000	71

Requerimiento 6

Índices creados

Índice	Tipo	Explicación
RESERVAS.(IDRESERVA, FECHAENTRADA)	Secundario y compuesto	Es índice es utilizado para las consultas relacionadas con la ocupación, esto se debe a que facilita la búsqueda de los datos durante la consulta por medio del filtro de la información a partir del id de cada reserva y la fecha de ingreso al hotel. Esta índice es secundario considerando que no se realiza sobre la primary key de la relación de reservas y es compuesto considerando que incluye dos columnas de la relación reservas.
CONSUMOS(IDSERVICIO, IDENTRADA)	Secundario y compuesto	Este índice es creado para la consulta que devuelve los mayores ingresos. Este índice hace que la realización de los Inner join se logren realizar de una manera más eficiente considerando que el índice esta creado a partir del id de cada servicio y el id de entrada de cada uno de los huéspedes. Considerando que esta consulta incluye continuamente el uso de estos datos, este índice permite que la consulta se logre realizar a un menor costo. Este índice es secundario considerando a que no se desarrollo sobre el primary key de la relación de

		consumo y es compuesto considerando a que incluye dos columnas.
--	--	---

Sentencia SQL

FECHAS CON MAYOR OCUPACION

```
SELECT FECHAENTRADA, COUNT (*) AS OCUPACION
FROM RESERVAS
GROUP BY FECHAENTRADA
ORDER BY OCUPACION DESC
FETCH FIRST 5 ROW ONLY;
```

FECHAS CON MENOR OCUPACION

```
SELECT FECHAENTRADA, COUNT (*) AS OCUPACION
FROM RESERVAS
GROUP BY FECHAENTRADA
ORDER BY OCUPACION ASC
FETCH FIRST 5 ROW ONLY;
```

FECHAS CON MAYORES INGRESOS

```
SELECT FECHAENTRADA, SUM(COSTOSERVICIO) AS INGRESOS
FROM CONSUMOS
INNER JOIN SERVICIOS ON CONSUMOS.IDSERVICIO=SERVICIOS.IDSERVICIO
INNER JOIN ENTRADAS ON ENTRADAS.IDENTRADA=CONSUMOS.IDENTRADA
INNER JOIN RESERVAS ON RESERVAS.IDRESERVA=ENTRADAS.IDRESERVA
GROUP BY FECHAENTRADA
ORDER BY INGRESOS DESC
FETCH FIRST 5 ROW ONLY;
```

Distribución de datos: Para poder cumplir con los objetivos de la consulta la cual tiene como objetivo lograr analizar la operación de Hotelandes, consideramos pertinente dividir el resultado según los requerimientos, los cuales pedían: fechas de los días con mayor ocupación, fechas de mayores ingreso y fechas de menor demanda. Para el caso de aquellas consultadas relacionadas con la ocupación, utilizamos únicamente la relación de RESERVAS la cual, según la fecha de entrada de los usuarios, se llevaba un registro con el objetivo de lograr comparar todas las fechas y lograr obtener aquellas donde existieran menos y más clientes hospedándose en el hotel. Para lograr hallar las fechas con mayores ingresos, se realizó la consulta a partir de la relación de CONSUMO, sin embargo, se realizaron diferentes INNER JOINS, con el objetivo de obtener información pertinente como el costo de cada uno de los SERVICIOS ofrecidos en el hotel y la fecha en la se hospedo el cliente. Para calcular la cantidad de ingresos que recibió el hotel, utilizamos la función de SUM para poder incluir todos los consumos realizados en un día en específico. Considerando que el enunciado no especifica la cantidad de registros que se deberían mostrar, en todos los casos se muestran 5 registros.

Plan de ejecución Oracle: Fechas con mayor ocupación

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT				5	100
SORT		ORDER BY		5	100
VIEW	SYS.null			5	99
Filter Predicates	from \$subquery\$002.rowlimit_\$\$_rownumber <= 5				
WINDOW		SORT PUSHED RANK		366	99
Filter Predicates	ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC) <= 5				
HASH		GROUP BY		366	99
TABLE ACCESS	RESERVAS	FULL		50000	94
Other XML					
(info)					

Fechas con menor ocupación

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT				5	100
SORT		ORDER BY		5	100
VIEW	SYS.null			5	99
Filter Predicates	from \$subquery\$002.rowlimit_\$\$_rownumber <= 5				
WINDOW		SORT PUSHED RANK		366	99
Filter Predicates	ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC) <= 5				
HASH		GROUP BY		366	99
TABLE ACCESS	RESERVAS	FULL		50000	94
Other XML					
(info)					

Fechas con mayores ingresos

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT				5	226
SORT		ORDER BY		5	226
VIEW	SYS.null			5	225
Filter Predicates	from \$subquery\$008.rowlimit_\$\$_rownumber <= 5				
WINDOW		SORT PUSHED RANK		366	225
Filter Predicates	ROW_NUMBER() OVER (ORDER BY SUM(SERVICIOS.COSTOSERVICIO) DESC) <= 5				
HASH		GROUP BY		366	225
HASH JOIN				100000	218
Access Predicates	CONSUMOS.IDSERVICIO = SERVICIOS.IDSERVICIO				
TABLE ACCESS	SERVICIOS	FULL		100	4
HASH JOIN				100000	214
Access Predicates	ENTRADAS.IDENTRADA = CONSUMOS.IDENTRADA				
HASH JOIN				50000	142
Access Predicates	RESERVAS.IDRESERVA = ENTRADAS.IDRESERVA				
TABLE ACCESS	ENTRADAS	FULL		50000	47
TABLE ACCESS	RESERVAS	FULL		50000	94
TABLE ACCESS	CONSUMOS	FULL		100000	71
Other XML					

Tiempos:

Fechas con mayor ocupación: 0,036 segundos

Fechas con menor ocupación: 0,055 segundos

Fechas con mayores ingresos: 0,112 segundos

Mejora con índice creado: A partir de los índices creados se vio una disminución considerable de los costos asociados dichas consultas. Esto se debe a que los índices permiten acceder con mayor facilidad a la información lo cual hace que la consulta sea mucho más eficiente. Entre los resultados de los costos tanto para las consultas relacionadas con las fechas con mayor y menor ocupación la disminución de los costos fue de 57 y para el caso de la consulta de fechas con mayores ingresos se logro reducir los costos con 77. Los beneficios de los índices creados para estas consultas también se vieron reflejados en los tiempos de ejecución de las consultas.

Nuevos tiempos

Fechas con mayor ocupación: 0,020 segundos

Fechas con menor ocupación: 0,023 segundos

Fechas con mayores ingresos: 0,087 segundos

Fechas con mayor ocupación:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT				5	43
SORT		ORDER BY		5	43
VIEW	SYS.NULL			5	42
Filter Predicates					
from \$subquery\$ _002.rowlimit_\$\$_rownumber <= 5					
WINDOW		SORT PUSHED RANK		366	42
Filter Predicates					
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC) <= 5					
HASH		GROUP BY		366	42
INDEX	RESERVAS_INDICE_1	FAST FULL SCAN		50000	37
Other XML					
{info}					
info type="has_user_tab"					
yes					

Fechas con menor ocupación:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT				5	43
SORT		ORDER BY		5	43
VIEW	SYS.NULL			5	42
Filter Predicates					
from \$subquery\$ _002.rowlimit_\$\$_rownumber <= 5					
WINDOW		SORT PUSHED RANK		366	42
Filter Predicates					
ROW_NUMBER() OVER (ORDER BY COUNT(*)) <= 5					
HASH		GROUP BY		366	42
INDEX	RESERVAS_INDICE_1	FAST FULL SCAN		50000	37
Other XML					
{info}					
info type="has user tab"					

Fechas con mayores ingresos:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT				5	149
SORT		ORDER BY		5	149
VIEW	SYS.NULL			5	148
Filter Predicates					
from \$subquery\$ _008.rowlimit_\$\$_rownumber <= 5					
WINDOW		SORT PUSHED RANK		366	148
Filter Predicates					
ROW_NUMBER() OVER (ORDER BY SUM(SERVICIOS.COSTOSERVICIO) DESC) <= 5					
HASH		GROUP BY		366	148
HASH JOIN			100000		141
Access Predicates					
CONSUMOS.IDSERVICIO=SERVICIOS.IDSERVICIO					
TABLE ACCESS	SERVICIOS	FULL		100	4
HASH JOIN			100000		137
Access Predicates					
ENTRADAS.IDENTRADA=CONSUMOS.IDENTRADA					
HASH JOIN			50000		85
Access Predicates					
RESERVAS.IDRESERVA=ENTRADAS.IDRESERVA					
TABLE ACCESS	ENTRADAS	FULL		50000	47
INDEX	RESERVAS_INDICE_1	FAST FULL SCAN		50000	37
INDEX	CONSUMO_INDICE	FAST FULL SCAN		100000	51
Other XML					
{info}					

Requerimiento 7

Índices creados

Índice	Tipo	Explicación
SERVICIOS.(IDSERVICIO, COSTOSERVICIO)	Secundario y compuesto	El índice creado sobre esta consulta facilita la búsqueda de los servicios según el id de cada uno de estos, junto con el costo de cada uno de estos. El índice es secundario considerando que no incluye el primary key y de igual forma es compuesto considerando que el índice es creado a partir de dos columnas.

Sentencia SQL

```
SELECT DISTINCT RESERVAS.IDUSUARIO,SUM(SERVICIOS.COSTOSERVICIO) AS  
COSTO_CONSUMO, SUM(TO_DATE(FECHASALIDA, 'MM/DD/YYYY')-  
TO_DATE(FECHAENTRADA, 'MM/DD/YYYY')+2) AS DIAS  
FROM CONSUMOS  
NATURAL JOIN SERVICIOS  
NATURAL JOIN ENTRADAS  
INNER JOIN RESERVAS ON ENTRADAS.IDRESERVA=RESERVAS.IDRESERVA  
GROUP BY RESERVAS.IDUSUARIO  
HAVING SUM(SERVICIOS.COSTOSERVICIO)>15000 OR  
SUM(TO_DATE(FECHASALIDA, 'MM/DD/YYYY')-TO_DATE(FECHAENTRADA,  
'MM/DD/YYYY')+2) >14;
```

Distribución de datos: Considerando que esta consulta tiene como objetivo lograr encontrar los buenos clientes, se deben verificar dos condiciones que harían a los individuos que se hospedan en el hotel andes unos buenos clientes. Entre las condiciones, esto incluye las personas que en un año se han hospedado por lo menos 2 semanas o aquellas personas cuyo consumo es mas de \$15000. Para realizar esta consulta, se tuvo en cuenta los CONSUMOS realizados por cada usuario y de igual se tenía que tener en cuenta tanto la FECHAENTRADA y FECHASALIDA, para cada vez que un cliente se hospedo en el hotel, esto con el objetivo de lograr calcular la cantidad de días que estuvo en el hotel. Esta consulta consistió en la realización de joins entre diferentes relaciones que nos permitieran agrupar los datos de tanto la cantidad de días hospedados en el hotel como los consumos por cada usuario según el id del usuario.

Plan de ejecución Oracle:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				1786 222
FILTER				
Filter Predicates				
OR				
SUM(SERVICIOS.COSTOSERVICIO)>15000				
SUM(TO_DATE(RESERVAS.FECHASALIDA,'MM/DD/YYYY')-TO_DATE(RESERVAS.FECHAENTRADA,'MM/DD/YYYY')+2)>14				
HASH		GROUP BY		1786 222
HASH JOIN			100000	218
Access Predicates				
CONSUMOS.IDSERVICIO=SERVICIOS.IDSERVICIO				
TABLE ACCESS	SERVICIOS	FULL	100	4
HASH JOIN			100000	214
Access Predicates				
CONSUMOS.IDENTRADA=ENTRADAS.IDENTRADA				
TABLE ACCESS	CONSUMOS	FULL	100000	71
HASH JOIN			50000	142
Access Predicates				
ENTRADAS.IDRESERVA=RESERVAS.IDRESERVA				
NESTED LOOPS				50000 142
NESTED LOOPS				
STATISTICS COLLECT				
TABLE ACCESS	ENTRADAS	FULL	50000	47
INDEX	RESERVAS_PK	UNIQUE SCAN		
Access Predicates				
ENTRADAS.IDRESERVA=RESERVAS.IDRESERVA				
TABLE ACCESS	RESERVAS	BY INDEX ROWID	1	94
TABLE ACCESS	RESERVAS	FULL	50000	94

Tiempos: 0,085 segundos

Mejora con índice creado: Con el objetivo de mejorar el costo de la consulta, considerando que cada vez que se ejecuta la consulta se realiza un recorrido total por todos los datos, una de las soluciones es crear un índice con el objetivo de realizar la consulta de una forma mas eficiente. El índice permitió que la consulta no tuviera que realizar un recorrido por todos los servicios y sus respectivos costo, esto disminuyo el tiempo de ejecución a 0,065 segundos. El motivo por el cual el costo no se reduce en una cantidad considerable es debido a la cantidad de datos que estan siendo incluidos en dicha consulta.

SELECT STATEMENT				1786 219
FILTER				
Filter Predicates				
OR				
SUM(SERVICIOS.COSTOSERVICIO)>15000				
SUM(TO_DATE(RESERVAS.FECHASALIDA,'MM/DD/YYYY')-TO_DATE(RESERVAS.FECHAENTRADA,'MM/DD/YYYY')+2)>14				
HASH		GROUP BY		1786 219
HASH JOIN			100000	215
Access Predicates				
CONSUMOS.IDSERVICIO=SERVICIOS.IDSERVICIO				
INDEX	SERVICIOS_INDICE	FULL SCAN	100	1
HASH JOIN			100000	214
Access Predicates				
CONSUMOS.IDENTRADA=ENTRADAS.IDENTRADA				
TABLE ACCESS	CONSUMOS	FULL	100000	71
HASH JOIN			50000	142
Access Predicates				
ENTRADAS.IDRESERVA=RESERVAS.IDRESERVA				
NESTED LOOPS				50000 142
NESTED LOOPS				
STATISTICS COLLECT				
TABLE ACCESS	ENTRADAS	FULL	50000	47
INDEX	RESERVAS_PK	UNIQUE SCAN		
Access Predicates				
ENTRADAS.IDRESERVA=RESERVAS.IDRESERVA				
TABLE ACCESS	RESERVAS	BY INDEX ROWID	1	94
TABLE ACCESS	RESERVAS	FULL	50000	94

Requerimiento 8

Índices creados

Índice	Tipo	Explicación
Reservas servicio(idservicio, fecha)	Compuesto	En este caso se crea el índice compuesto para que sea más eficiente para que la agrupación y ordenamiento sea más rápido.

Sentencia SQL

```
SELECT
  r.IDSERVICIO,
  TO_CHAR(TO_DATE(r.FECHA, 'MM/DD/YYYY'), 'IW') AS SEMANA,
  TO_CHAR(TO_DATE(r.FECHA, 'MM/DD/YYYY'), 'YYYY') AS ANIO ,
  COUNT(r.IDRESERVASERVICIO) AS NUMERODERESERVAS
FROM RESERVASSERVICIO r
GROUP BY r.IDSERVICIO, TO_CHAR(TO_DATE(r.FECHA, 'MM/DD/YYYY'), 'IW'),
         TO_CHAR(TO_DATE(r.FECHA, 'MM/DD/YYYY'), 'YYYY')
HAVING COUNT(r.IDRESERVASERVICIO) < 10
ORDER BY TO_CHAR(TO_DATE(r.FECHA, 'MM/DD/YYYY'), 'YYYY'),
         TO_CHAR(TO_DATE(r.FECHA, 'MM/DD/YYYY'), 'IW');
```

Distribución de datos:

En la distribución de datos de esta consulta se quiere obtener los servicios que en alguna semana tuvieron menos de 3 reservas, no obstante, en la base de datos no se encuentra uno menor a 3. Por lo tanto, se tomó menor a 10 servicios. Para esto se cuentan el número de reservas en el año 2024 y se hacen las respectivas condiciones.

Plan de ejecución Oracle:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1295	102
SORT		ORDER BY	1295	102
FILTER				
Filter Predicates				
COUNT(*) < 10				
HASH		GROUP BY	1295	102
TABLE ACCESS	RESERVASSERVICIO	FULL	100000	95

Tiempos:

Para esta consulta se demora 0,14 segundos

Mejora con índice creado:

Se puede ver que el costo total baja ya que en vez de hacer un acceso completo a la tabla se realiza un escaneo rápido con el cual se obtiene un costo de 74.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1295	74
SORT		ORDER BY	1295	74
FILTER				
Filter Predicates				
COUNT(*) < 10				
HASH		GROUP BY	1295	74
INDEX	IDX_RESERVASSERVICIO_ID...	FAST FULL SCAN	100000	66

Requerimiento 9

Índices creados

Índice	Tipo	Explicación
CONSUMOS(idservicio, nHabitacion)	Secundario y compuesto.	En este caso se tiene un índice compuesto en la tabla de consumos entre las columnas de idservicio y nHabitación, con esto se facilita el acceso a cada uno de los servicios consumidos por las habitaciones, por lo que

		la búsqueda de la habitación y de los servicios que son consumidos por estas habitaciones es más eficiente. Este es un índice secundario debido a que no está exclusivamente en la llave primaria de la tabla, además es compuesto debido a que no se encuentra solo en una de las columnas de la tabla.
--	--	--

Sentencia SQL:

```
SELECT u.nombre, COUNT(u.idusuario)
FROM reservas r
INNER JOIN usuarios u ON (r.idusuario = u.idusuario)
INNER JOIN consumos c ON(c.nHabitacion = r.nHabitacion)
INNER JOIN reservasservicio rs ON(rs.nHabitacion = r.nHabitacion)
WHERE c.idservicio = 10
AND rs.idservicio = 10
AND (rs.fecha < '02/05/2024' and rs.fecha > '01/05/2024')
GROUP BY u.nombre
ORDER BY COUNT(u.idusuario) DESC;
```

Distribución de datos:

Para este requerimiento se tienen los datos que se usan en 4 tablas diferentes, se tiene en la tabla RESERVAS, CONSUMOS y RESERVASSERVICIO. En este caso se debe asignar una de las habitaciones a uno de los usuarios en específico y ver si este usuario consumió más de una vez un servicio en un tiempo determinado. Para esto se hace un JOIN entre estas cuatro tablas teniendo en cuenta los campos que tienen valores iguales para cada uno de los JOINS.

Plan de ejecución Oracle:

SELECT STATEMENT			144	309
SORT		ORDER BY	144	309
HASH		GROUP BY	144	309
HASH JOIN			24879	308
Access Predicates				
C.NHABITACION=R.NHABITACION				
TABLE ACCESS	CONSUMOS	FULL	1012	71
Filter Predicates				
C.IDSERVICIO=10				
HASH JOIN			9834	237
Access Predicates				
R.IDUSUARIO=ITEM_1				
HASH JOIN			9834	190
Access Predicates				
RS.NHABITACION=R.NHABITACION				
TABLE ACCESS	RESERVASSERVICIO	FULL	79	95
Filter Predicates				
AND				
RS.IDSERVICIO=10				
RS.FECHA<'02/05/2024'				
RS.FECHA>'01/05/2024'				
TABLE ACCESS	RESERVAS	FULL	50000	95
VIEW	SYS.VW_GBF_35		20000	47
TABLE ACCESS	USUARIOS	FULL	20000	47

En este caso se usan varios HASH JOIN para unir cada una de las tablas y poder acceder a datos específicos en la consulta, por ejemplo, el id del servicio. De igual manera, se puede ver que el

acceso de todas las tablas es un acceso FULL lo que quiere decir que se leen todos los datos lo que conlleva a que la consulta tenga un costo alto.

Tiempos:

Prueba 1: 0.24

Prueba 2: 0.28

Prueba 3: 0.38

Mejora con índice creado:

SELECT STATEMENT			144	242
SORT	ORDER BY		144	242
HASH	GROUP BY		144	242
HASH JOIN			24879	240
Access Predicates				
C.NHABITACION=R.NHABITACION				
INDEX	SERVHABIT	RANGE SCAN	1012	3
Access Predicates				
C.IDSERVICIO=10				
HASH JOIN			9834	237
Access Predicates				
R.IDUSUARIO=ITEM_1				
HASH JOIN			9834	190
Access Predicates				
RS.NHABITACION=R.NHABITACION				
TABLE ACCESS	RESERVASSERVICIO	FULL	79	95
Filter Predicates				
AND				
RS.IDSERVICIO=10				
RS.FECHA<'02/05/2024'				
RS.FECHA>'01/05/2024'				
TABLE ACCESS	RESERVAS	FULL	50000	95
VIEW	SYS.VW_GBF_35		20000	47
TABLE ACCESS	USUARIOS	FULL	20000	47

Con el índice creado se puede ver una reducción del costos del 21,68% lo que quiere decir que este índice creado ayuda a que se optimicen los costos, más específicamente cuando se esta accediendo al servicio específico ya no se hace una lectura completa de todos los datos, sino que se hace solo una búsqueda por medio

Requerimiento 10

Índices creados

Índice	Tipo	Explicación
Consumos.(nHabitación, idservicio)	Compuesto	Se crea el índice compuesto para hacer primero la unión entre nhabitacion y luego el de idservicio

Sentencia SQL:

```
SELECT u.idusuario, u.nombre, u.email
FROM usuarios u
WHERE u.idusuario NOT IN (
  SELECT r.idusuario
  FROM consumos c
  JOIN reservas r ON r.nHabitacion = c.nHabitacion
  JOIN servicios s ON s.idservicio = c.idservicio
  JOIN reservasservicio rs ON rs.idservicio = s.idservicio
  WHERE s.idservicio = :id AND (rs.fecha > :inicio AND rs.fecha < :fin));
```

Distribución de datos:

En esta consulta se quieren obtener los datos de los clientes que no han consumido un servicio en unas fechas específicas, para esto se hace la exclusión del requerimiento 9 y se obtienen los datos del usuario.

Plan de ejecución Oracle:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1690	309
HASH JOIN		ANTI	1690	309
Access Predicates U.IDUSUARIO=IDUSUARIO				
TABLE ACCESS	USUARIOS	FULL	20000	47
VIEW	SYS.VW_NSO_1		125000	261
FILTER				
Filter Predicates :FIN>:INICIO				
HASH JOIN			125000	261
Access Predicates R.NHABITACION=C.NHABITACION				
HASH JOIN		RIGHT SEMI	1000	166
Access Predicates RS.IDSERVICIO=C.IDSERVICIO				
TABLE ACCESS	RESERVASSERVICIO	FULL	3	95
Filter Predicates AND RS.FECHA>:INICIO RS.FECHA<:FIN RS.IDSERVICIO=TO_NUMBER(:ID)				
TABLE ACCESS	CONSUMOS	FULL	1000	71
Filter Predicates C.IDSERVICIO=TO_NUMBER(:ID)				
TABLE ACCESS	RESERVAS	FULL	50000	95

Tiempos:

La consulta se demora 0,111 segundos para los siguientes valores:

Id del usuario = 20

Fecha de inicio = 10/10/2024

Fecha de fin = 12/12/2024

Mejora con índice creado:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1690	287
HASH JOIN		ANTI	1690	287
Access Predicates				
U.IDUSUARIO=IDUSUARIO				
TABLE ACCESS	USUARIOS	FULL	20000	47
VIEW	SYS.VW_NSO_1		125000	239
FILTER				
Filter Predicates				
:FIN>:INICIO				
HASH JOIN			125000	239
Access Predicates				
R.NHABITACION=C.NHABITACION				
HASH JOIN		RIGHT SEMI	1000	144
Access Predicates				
RS.IDSERVICIO=C.IDSERVICIO				
TABLE ACCESS	RESERVASSERVICIO	FULL	3	95
Filter Predicates				
AND				
RS.FECHA>:INICIO				
RS.FECHA<:FIN				
RS.IDSERVICIO=TO_NUMBER(:ID)				
INDEX	IDX_CONSUMOS_NHABITA...	FAST FULL SCAN	1000	49
Filter Predicates				
C.IDSERVICIO=TO_NUMBER(:ID)				
TABLE ACCESS	RESERVAS	FULL	50000	95

Requerimiento 11

Índices creados

Índice	Tipo	Explicación
SERVICIOS.(IDSERVICIO, COSTOSERVICIO)	Secundario y compuesto	El índice creado sobre esta consulta facilita la búsqueda de los servicios según el id de cada uno de estos, junto con el costo de cada uno de estos. El índice es secundario considerando que no incluye el primary key y de igual forma es compuesto considerando que el índice es creado a partir de dos columnas.
RESERVAS.(FECHAENTRADA, NHABITACION)	Secundario y compuesto	El índice en es creado considerando que esto facilitaría la búsqueda de los datos considerando que las columnas de fecha de entrada y el número de habitación es utilizada constante en la consulta. Este índice es secundario considerando que no incluye a el primary key de la relación de reservas y es secundario considerando a que utiliza la información de dos columnas.

Sentencia SQL

SERVICIO CON MAYOR CANTIDAD DE SOLICITUDES

```

SELECT TO_CHAR(TO_DATE(FECHAENTRADA, 'MM/DD/YYYY'), 'WW') AS SEMANA,
S.IDSERVICIO , COUNT(S.IDSERVICIO) AS SOLICITUDES_SERVICIO,
SUM(COSTOSERVICIO) AS CONSUMO
FROM RESERVAS R
INNER JOIN ENTRADAS E ON R.IDRESERVA=E.IDRESERVA
INNER JOIN CONSUMOS C ON C.IDENTRADA=E.IDENTRADA
INNER JOIN SERVICIOS S ON S.IDSERVICIO=C.IDSERVICIO
WHERE TO_CHAR(TO_DATE(FECHAENTRADA, 'MM/DD/YYYY'), 'WW')=1
GROUP BY TO_CHAR(TO_DATE(FECHAENTRADA, 'MM/DD/YYYY'), 'WW'),
S.IDSERVICIO
ORDER BY COUNT(S.IDSERVICIO) DESC
FETCH FIRST 1 ROW ONLY;

```

SERVICIO CON MENOR CANTIDAD DE SOLICITUDES

```

SELECT TO_CHAR(TO_DATE(FECHAENTRADA, 'MM/DD/YYYY'), 'WW') AS SEMANA,
S.IDSERVICIO , COUNT(S.IDSERVICIO) AS SOLICITUDES_SERVICIO,
SUM(COSTOSERVICIO) AS CONSUMO
FROM RESERVAS R
INNER JOIN ENTRADAS E ON R.IDRESERVA=E.IDRESERVA
INNER JOIN CONSUMOS C ON C.IDENTRADA=E.IDENTRADA
INNER JOIN SERVICIOS S ON S.IDSERVICIO=C.IDSERVICIO
WHERE TO_CHAR(TO_DATE(FECHAENTRADA, 'MM/DD/YYYY'), 'WW')=1
GROUP BY TO_CHAR(TO_DATE(FECHAENTRADA, 'MM/DD/YYYY'), 'WW'),
S.IDSERVICIO
ORDER BY COUNT(S.IDSERVICIO) ASC
FETCH FIRST 1 ROW ONLY;

```

HABITACION CON MAYOR CANTIDAD DE SOLICITUDES

```

SELECT TO_CHAR(TO_DATE(FECHAENTRADA, 'MM/DD/YYYY'), 'WW') AS SEMANA,
R.NHABITACION, COUNT(R.NHABITACION) AS CANTIDAD_HABITACION
FROM RESERVAS R
WHERE TO_CHAR(TO_DATE(FECHAENTRADA, 'MM/DD/YYYY'), 'WW')=52
GROUP BY TO_CHAR(TO_DATE(FECHAENTRADA, 'MM/DD/YYYY'), 'WW'),
R.NHABITACION
ORDER BY COUNT(R.NHABITACION) DESC
FETCH FIRST 1 ROW ONLY;

```

HABITACION CON MENOR CANTIDAD DE SOLICITUDES

```

SELECT TO_CHAR(TO_DATE(FECHAENTRADA, 'MM/DD/YYYY'), 'WW') AS SEMANA,
R.NHABITACION, COUNT(R.NHABITACION) AS CANTIDAD_HABITACION
FROM RESERVAS R
WHERE TO_CHAR(TO_DATE(FECHAENTRADA, 'MM/DD/YYYY'), 'WW')=52
GROUP BY TO_CHAR(TO_DATE(FECHAENTRADA, 'MM/DD/YYYY'), 'WW'),
R.NHABITACION

```

ORDER BY COUNT(R.NHABITACION) ASC
FETCH FIRST 1 ROW ONLY;

Distribución de datos: Considerando que el objetivo de esta consulta avanzada es poder realizar una consulta del funcionamiento del hotel, se incluyo las operaciones pertinentes para poder hallar el servicio mas y menos consumido según una semana en específico y de igual forma la habitación mas y menos solicitada para cada semana. Con el objetivo de poder realizar una consulta mucho mas limpia y entendible para el gerente, esta consulta avanzada incluye un espacio para poder ingresar como parámetro la semana cuyos datos se desean consultar. Para el caso de las consultas interesadas en información acerca de los servicios mas solicitados por parte de los clientes, se implementaron varios INNER JOINS que permitieron relacionar información como la FECHAENTRADA, COSTOSERVICIO Y el IDSERVICIO. Esto con el objetivo de determinar en primer lugar, la semana en la que se obtuvo el servicio , el tipo de servicio y de igual forma el costo de dicho servicio. Para el caso de las consultas asociadas a las habitaciones se tomó únicamente información de la relación de reservas, a partir de los datos como la FECHAENTRA se calculó la semana en la que un cliente se hospedo en dicha habitación y a partir de NHABITACION se calculó la cantidad de veces en cada semana donde esa habitación fue solicitada.

Plan de ejecución Oracle:

Servicio con mayor cantidad de solicitudes

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				
SORT				221
VIEW	SYS.NULL	ORDER BY		221
Filter Predicates				220
from\$_subquery\$_008.rowlimit_\$\$rownumber <= 1				
WINDOW		SORT PUSHED RANK		1000
Filter Predicates				220
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC) <= 1				
HASH		GROUP BY		1000
HASH JOIN				220
Access Predicates				218
S.IDSERVICIO=C.IDSERVICIO				
TABLE ACCESS	SERVICIOS	FULL		100
HASH JOIN				1000
Access Predicates				
C.IDENTRADA=E.IDENTRADA				
HASH JOIN				500
Access Predicates				
R.IDRESERVA=E.IDRESERVA				
TABLE ACCESS	RESERVAS	FULL		500
Filter Predicates				
TO_NUMBER(TO_CHAR(TO_DATE(R.FECHAENTRADA,'MM/DD/YYYY'),'VW'))=1				
TABLE ACCESS	ENTRADAS	FULL		50000
TABLE ACCESS	CONSUMOS	FULL		100000

Servicio con menor cantidad de solicitudes

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				221
SORT		ORDER BY		221
VIEW	SYS.null			220
Filter Predicates				
from\$_subquery\$_008.rowlimit_\$\$_rownumber <= 1				
WINDOW		SORT PUSHED RANK		220
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*)) <= 1				
HASH		GROUP BY		220
HASH JOIN				218
Access Predicates				
S.IDSERVICIO=C.IDSERVICIO				
TABLE ACCESS	SERVICIOS	FULL		4
HASH JOIN				214
Access Predicates				
C.IDENTRADA=E.IDENTRADA				
HASH JOIN				143
Access Predicates				
R.IDRESERVA=E.IDRESERVA				
TABLE ACCESS	RESERVAS	FULL		95
Filter Predicates				
TO_NUMBER(TO_CHAR(TO_DATE(R.FECHAENTRADA,'MM/DD/YYYY'),'WW'))=1				
TABLE ACCESS	ENTRADAS	FULL		47
TABLE ACCESS	CONSUMOS	FULL		71

Habitación con mayor cantidad de solicitudes

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				98
SORT		ORDER BY		98
VIEW	SYS.null			97
Filter Predicates				
from\$_subquery\$_002.rowlimit_\$\$_rownumber <= 1				
WINDOW		SORT PUSHED RANK		97
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC) <= 1				
HASH		GROUP BY		97
TABLE ACCESS	RESERVAS	FULL		95
Filter Predicates				
TO_NUMBER(TO_CHAR(TO_DATE(FECHAENTRADA,'MM/DD/YYYY'),'WW'))=52				

Habitación con menor cantidad de solicitudes

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				98
SORT		ORDER BY		98
VIEW	SYS.null			97
Filter Predicates				
from\$_subquery\$_002.rowlimit_\$\$_rownumber <= 1				
WINDOW		SORT PUSHED RANK		97
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*)) <= 1				
HASH		GROUP BY		97
TABLE ACCESS	RESERVAS	FULL		95
Filter Predicates				
TO_NUMBER(TO_CHAR(TO_DATE(FECHAENTRADA,'MM/DD/YYYY'),'WW'))=52				

Tiempos:

Servicio con mayor cantidad de solicitudes: 0,094 segundos

Servicio con menor cantidad de solicitudes: 0,09 segundos

Habitación con mayor cantidad de solicitudes: 0,063 segundos

Habitación con menor cantidad de solicitudes: 0,055 segundos

Mejora con índice creado: Al realizar los índices, se puede ver una reducción en los costos asociados con la lectura de los datos. Para el caso de las consultas relacionadas con las habitaciones, se puede ver una reducción en los costos 58, es debe a que el índice creado a partir de la fecha de entrada y el número de habitación hace mucho más eficiente esta consulta. De igual, forma los tiempos de ejecución para todos los casos se reducen significativamente cuando se usan los índices.

Servicio con mayor cantidad de solicitudes:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT				1	218
SORT		ORDER BY		1	218
VIEW	SYS.NULL			1	217
Filter Predicates					
from\$_subquery\$_008.rowlimit_\$\$_rownumber <= 1					
WINDOW		SORT PUSHED RANK		1000	217
Filter Predicates					
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC) <= 1					
HASH		GROUP BY		1000	217
HASH JOIN				1000	215
Access Predicates					
S.IDSERVICIO=C.IDSERVICIO					
INDEX	SERVICIOS_INDICE	FULL SCAN		100	1
HASH JOIN				1000	214
Access Predicates					
C.IDENTRADA=E.IDENTRADA					
HASH JOIN				500	143
Access Predicates					
R.IDRESERVA=E.IDRESERVA					
TABLE ACCESS	RESERVAS	FULL		500	95
Filter Predicates					
TO_NUMBER(TO_CHAR(TO_DATE(R.FECHAENTRADA,'MM/DD/YYYY'),'WW'))=1					
TABLE ACCESS	ENTRADAS	FULL		50000	47
TABLE ACCESS	CONSUMOS	FULL		100000	71

Servicio con menor cantidad de solicitudes:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT				1	218
SORT		ORDER BY		1	218
VIEW	SYS.NULL			1	217
Filter Predicates					
from\$_subquery\$_008.rowlimit_\$\$_rownumber <= 1					
WINDOW		SORT PUSHED RANK		1000	217
Filter Predicates					
ROW_NUMBER() OVER (ORDER BY COUNT(*)) <= 1					
HASH		GROUP BY		1000	217
HASH JOIN				1000	215
Access Predicates					
S.IDSERVICIO=C.IDSERVICIO					
INDEX	SERVICIOS_INDICE	FULL SCAN		100	1
HASH JOIN				1000	214
Access Predicates					
C.IDENTRADA=E.IDENTRADA					
HASH JOIN				500	143
Access Predicates					
R.IDRESERVA=E.IDRESERVA					
TABLE ACCESS	RESERVAS	FULL		500	95
Filter Predicates					
TO_NUMBER(TO_CHAR(TO_DATE(R.FECHAENTRADA,'MM/DD/YYYY'),'WW'))=1					
TABLE ACCESS	ENTRADAS	FULL		50000	47
TABLE ACCESS	CONSUMOS	FULL		100000	71

Habitación con mayor cantidad de solicitudes:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT				1	40
SORT		ORDER BY		1	40
VIEW	SYS.NULL			1	39
Filter Predicates					
from\$_subquery\$_002.rowlimit_\$\$_rownumber <= 1					
WINDOW		SORT PUSHED RANK		500	39
Filter Predicates					
ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC) <= 1					
HASH		GROUP BY		500	39
INDEX	RESERVAS_INDICE1	FAST FULL SCAN		500	37
Filter Predicates					
TO_NUMBER(TO_CHAR(TO_DATE(FECHAENTRADA,'MM/DD/YYYY'),'WW'))=52					

Habitación con menor cantidad de solicitudes:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				
SORT				
VIEW	SYS.NULL			
Filter Predicates				
from \$subquery\$_002.rowlimit_\$\$_rownumber <= 1				
WINDOW				
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY COUNT(*) <= 1				
HASH				
INDEX	RESERVAS_INDICE1			
Filter Predicates				
TO_NUMBER(TO_CHAR(TO_DATE(FECHAENTRADA,'MM/DD/YYYY'),'WW'))=52				
Other XML				
Info				

Requerimiento 12

Índices creados

Índice	Tipo	Explicación
Entradas.idreservas	simple	Al hacer este índice se espera que al realizar el group by baje el costo que se muestra en el plan de ejecución. Además de esto se espera que al hacer el join sea más eficiente

Sentencia SQL:

```
SELECT r.IDUSUARIO AS id_cliente,
       TO_CHAR(TO_DATE(r.FECHAENTRADA, 'MM/DD/YYYY'), 'YYYY') AS year,
       COUNT(DISTINCT TO_CHAR(TO_DATE(r.FECHAENTRADA, 'MM/DD/YYYY'), 'Q'))
AS distintos_trimestres,
       COUNT(*) AS numero_de_entradas
FROM RESERVAS r JOIN ENTRADAS e ON r.IDRESERVA = e.IDRESERVA
GROUP BY r.IDUSUARIO, TO_CHAR(TO_DATE(r.FECHAENTRADA,
'MM/DD/YYYY'), 'YYYY')
HAVING COUNT(DISTINCT TO_CHAR(TO_DATE(r.FECHAENTRADA,
'MM/DD/YYYY'), 'Q')) = 4
ORDER BY id_cliente, year;
```

Distribución de datos:

En esta consulta se quiere obtener los clientes excelentes por trimestre, para esto se consiguen los datos de los usuarios por trimestre y se hace un conteo para que sean igual a 4. Si son igual a 4 es porque ese id apareció en todos los trimestres.

Plan de ejecución Oracle:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			184	518
SORT		ORDER BY	184	518
FILTER				
Filter Predicates				
COUNT(ITEM_1)=4				
HASH		GROUP BY	184	518
VIEW	SYS.VW_DAG_0		50000	514
HASH		GROUP BY	50000	514
HASH JOIN			50000	142
Access Predicates				
R.IDRESERVA=E.IDRESERVA				
TABLE ACCESS	ENTRADAS	FULL	50000	47
TABLE ACCESS	RESERVAS	FULL	50000	94

Tiempos:

Para esta consulta se demora 0,149 segundos en ejecutarse

Mejora con índice creado:

Al hacer el índice se puede ver que se logra bajar el costo de ejecución del último requerimiento a 493.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			184	493
SORT		ORDER BY	184	493
FILTER				
Filter Predicates				
COUNT(ITEM_1)=4				
HASH		GROUP BY	184	493
VIEW	SYS.VW_DAG_0		50000	489
HASH		GROUP BY	50000	489
HASH JOIN			50000	117
Access Predicates				
R.IDRESERVA=E.IDRESERVA				
INDEX	IDX_ENTRADAS_IDRESERVA	FAST FULL SCAN	50000	22
TABLE ACCESS	RESERVAS	FULL	50000	94

REQUERIMIENTOS NO FUNCIONALES

Requerimiento 1:

Se realizaron diferentes pruebas y ninguna de las consultas tiene una duración mayor a 0.8 segundos y todas las consultas cuentan con tiempos aceptables. Con esto se cumple el requisito de la eficiencia además se tiene una base de datos de gran tamaño en donde las consultas son independientes del tamaño de la base.

Requerimiento 2:

Para los requerimientos en donde se tenían parámetros cambiantes se hicieron pruebas cambiando estos parámetros y se evidencio que el cambio de estos parámetros era eficiente y el tiempo de ejecución no cambiaba en gran medida cuando se tenían diferentes parámetros.

Requerimiento 3:

Se trata de que el esquema sea lo más eficiente por medio de la creación de índices que hagan más eficientes las consultas.

Carga de datos:

La carga de datos en la base de datos del hotel se llevó a cabo mediante un proceso de generación de tablas a partir de archivos de Excel, facilitando así la importación de datos a SQL Developer. En primer lugar, se crearon archivos de Excel que contenían la información detallada de las diferentes entidades de datos que necesitaban ser almacenadas en la base de datos del hotel. Estos archivos Excel sirvieron como fuentes de datos primarias y se organizaron de manera que cada

hoja de cálculo representaba una entidad o tabla específica, como "Clientes", "Habitaciones", "Reservas" y "Servicios" entre otras de las entidades necesarias para la operación del hotel. Estos archivos se pueden ver en el archivo BaseDeDatos.

Luego, se utilizó SQL Developer para importar estos archivos de Excel en la base de datos. Esto se realizó por medio de una funcionalidad de la aplicación la cual permite la importación de datos desde archivos de Excel. En SQL Developer, se mapearon las columnas de las hojas de cálculo de Excel a las columnas de las tablas de la base de datos, asegurando una correspondencia precisa de los datos. De esta manera, se generaron las tablas en la base de datos, y los datos se insertaron automáticamente en sus respectivas ubicaciones. Este enfoque simplificó significativamente el proceso de carga de datos y garantizó la integridad y coherencia de la información almacenada en la base de datos del hotel.

Además de la carga de datos a través de archivos de Excel, se implementaron funciones nativas de Excel para generar un volumen significativo de datos aleatorios. Estas funciones permitieron simular situaciones realistas y poblaron las tablas de la base de datos del hotel con información ficticia, pero representativa. Por ejemplo, se utilizaron funciones como RAND(), RANDBETWEEN(), para crear valores aleatorios de ocupación en diferentes habitaciones, duración de estancias, precios de servicios y otros datos relevantes para la operación del hotel.

En resumen, la carga de datos en la base de datos del hotel se llevó a cabo mediante la generación de tablas a partir de archivos de Excel, utilizando SQL Developer como una herramienta efectiva para importar y mapear los datos. Este enfoque proporcionó una solución eficiente y organizada para incorporar información crucial en la base de datos, permitiendo una gestión más efectiva de los datos relacionados con las operaciones del hotel.