Análisis de consultas

Requerimiento Funcional de Consulta 9

RFC9-Se quiere conocer la información de los clientes que consumieron al menos una vez un determinado servicio del hotel, en un rango de fechas. Los resultados deben ser clasificados según un criterio deseado por quien realiza la consulta. En la clasificación debe ofrecerse la posibilidad de agrupamiento y ordenamiento de las respuestas según los intereses del usuario que consulta como, por ejemplo, por los datos del cliente, por fecha y número de veces que se utilizó el servicio. Esta operación está disponible para el recepcionista del hotel, el gerente del hotel y también para los organizadores de eventos.

Sentencia SQL

```
Select us.nombre, us.identificacion, count(*)
from usuario us
inner join reserva res
on us.identificacion=res.id_usuario
inner join apartan ap
on res.id=ap.idreserva
inner join
habitacion hab
on ap.numerohabitacion=hab.numerohabitacion
inner join sirven sirv
on hab.numerohabitacion=sirv.numerohabitacion
where sirv.FECHAUSO between 2019010101 and 2019070101
and sirv.IDSERVICIO=1
group by us.identificacion, us.nombre;
```

Imagen (1)

Para solucionar el requerimiento funcional 9, se usó la sentencia SQL mostrada en la imagen 1. Se utilizaron las tablas usuario, reserva, apartan, habitaciones y

sirven, los cuales contenían la información necesaria para resolver el requerimiento funcional. La tabla usuario, contiene 9001 registros, la tabla reserva contiene 10000 registros, la tabla apartan contiene 53412 registros, la tabla habitación, contiene 100 registros y la tabla sirven contiene 200000 registros. A raíz de que la tabla sirven contiene una gran cantidad de registros, y dentro de la consulta hay una condición de búsqueda sobre la fecha de uso, se decidió crear un índice secundario, de un árbol B+, ya que este nos facilita las consultas en rangos.

	INDEX_OWNER	INDEX_NAME	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	DESCEND
1	ISIS2304A141910	FECHAUSO	ISIS2304A141910	SIRVEN	FECHAUSO	1	ASC

	Nombre	Valor
1	OWNER	ISIS2304A141910
2	INDEX_NAME	FECHAUSO
3	TABLE_OWNER	ISIS2304A141910
4	TABLE_NAME	SIRVEN
5	PARTITION_NAME	(null)
6	PARTITION_POSITION	(null)
7	SUBPARTITION_NAME	(null)
8	SUBPARTITION_POSITION	(null)
9	OBJECT_TYPE	INDEX
10	BLEVEL	2
11	LEAF_BLOCKS	503
12	DISTINCT_KEYS	8556
13	AVG_LEAF_BLOCKS_PER_KEY	1
14	AVG_DATA_BLOCKS_PER_KEY	22
15	CLUSTERING_FACTOR	195400
16	NUM_ROWS	200000
17	AVG_CACHED_BLOCKS	(null)
18	AVG_CACHE_HIT_RATIO	(null)
19	SAMPLE_SIZE	200000
20	LAST_ANALYZED	18/05/19
21	GLOBAL_STATS	YES
22	USER_STATS	NO
23	STATTYPE_LOCKED	(null)
24	STALE_STATS	NO
25	SCOPE	SHARED

Adicionalmente se realizaron los cálculos del índice para determinar si es viable y cabe en memoria: Se sabe que hay 503 bloques * 2400 que es lo que pesa un bloque, lo que es igual a 1207200 bytes lo que es muy inferior a la RAM.

Adicionalmente en la consulta se usó el índice de la llave primaria de servicio, ya que se debía hacer un index scan sobre el mismo para encontrar solo los que contienen el id del servicio solicitado.

Por otra parte, los valores usados en la consulta, son la fecha inicial del uso, que equivale a 2019010101 y la fecha final de uso que equivale a 2019070101 de un servicio con identificación 1.

Según los parámetros ingresados, el tiempo de ejecución de la sentencia es el siguiente:

Al cambiar los parámetros de consulta en diferentes ocasiones, aumentando el rango de búsqueda de las fechas y disminuyéndolo, se observó que el tiempo de ejecución de la sentencia no variaba y se mantenía constante, oscilaba entre los 0,02 y 0,06 segundos. Los datos de prueba son los siguientes.

2019010101	2019050101	3	0,034
2018050101	2019020101	5	0,055
2019030101	2019040101	9	0,035

El plan de consulta ofrecido por Oracle es el siguiente

SELECT STATEMENT			5981	241
HASH		GROUP BY	5981	241
HASH JOIN			53512	239
Access Predicates				
AP.NUMEROHABITACION=ITEM_1				
VIEW	SYS.VW_GBF_19		100	173
HASH		GROUP BY	100	173
TABLE ACCESS	SIRVEN	FULL	11176	171
Filter Predicates				
AND				
SIRV.IDSERVICIO=1				
SIRV.FECHAUSO<=20190701O1				
SIRV.FECHAUSO>=20190101O1				
HASH JOIN			53512	67
Access Predicates				
RES.ID=AP.IDRESERVA				
NESTED LOOPS			53512	67
STATISTICS COLLECTOR				
HASH JOIN			10000	39
Access Predicates				
US.IDENTIFICACION=RES.ID_USUARIO				
TABLE ACCESS	RESERVA	FULL	10000	17
TABLE ACCESS	USUARIO	FULL	9000	22
INDEX	APARTAN_PK	RANGE SCAN	5	42
Access Predicates				
RES.ID=AP.IDRESERVA				
TABLE ACCESS	APARTAN	FULL	53512	27
Other XML				

Primero hace un hash join entre las tablas reserva y usuario, debido a que ambas tablas no caben en memoria, y se busca encontrar las llaves que coincidan. Después, se hace un nested loop join, usando los índices de la tabla apartan, los cuales fueron encontrados mediante un index scan. La tabla resultante del nested loop join se le hace un hash join con la tabla o la vista resultante de la selección de las tuplas con el parámetro de fecha determinado.

Plan de consulta desarrollado

```
Select Statement
        Hash Join
                Usuario                                 Full Table Scan

                Hash Join
                        Reserva                         Full Table Scan

                        Nested Loop Join
                                Apartan                 Full Table Scan

                                Hash Join
                                        Sirven          Index Scan
                                                Filter Predicate on id=1, 2019010101<=FechaUso<=2019050101
                                        Habitacion      Full Table Scan
```

El plan de consulta desarrollado por el grupo de trabajo es muy similar al planteado por Oracle. En un principio, se hace un index scan a la tabla sirven, ya que se debe encontrar las tuplas con ServicioId igual a 1. Por otro lado, toca leer todas las tuplas de habitación, para poder el join. Se realiza un Hash join, debido, a que toca juntarlos por una llave igual, esto genera que las colisiones sean correctas. Seguido se hace un Nested Loop Join entre la tabla resultante del hash anterior y la tabla apartan, la cual tiene que leerse completa en lectura secuencial rápida, cabe resaltar que al

realizar el Nested Loop Join, significa que una de las tablas cabe en memoria. Después se hacen dos hashes joins, uno con la tabla reservas y otro con la tabla usuarios.

Requerimiento Funcional de Consulta 10

RFC-10Se quiere conocer la información de los clientes que NO consumieron ninguna vez un determinado servicio del hotel, en un rango de fechas. Los resultados deben ser clasificados según un criterio deseado por quien realiza la consulta. En la clasificación debe ofrecerse la posibilidad de agrupamiento y ordenamiento de las respuestas según los intereses del usuario que consulta como, por ejemplo, por los datos del cliente, por fecha y número de veces que se utilizó el servicio. Esta operación está disponible para el recepcionista del hotel, el gerente del hotel y también para los organizadores de eventos.

Sentencia SQL

```
select *
from usuario
where identificacion not in(
Select us.identificacion
from usuario us
inner join reserva res
on us.identificacion=res.id_usuario
inner join apartan ap
on res.id=ap.idreserva
inner join
habitacion hab
on ap.numerohabitacion=hab.numerohabitacion
inner join sirven sirv
on hab.numerohabitacion=sirv.numerohabitacion
where sirv.FECHAUSO between 2019010101 and 20190501
and sirv.IDSERVICIO=1);
```

Para solucionar el requerimiento funcional de consulta 10, se tuvo en cuenta el requerimiento anterior, debido a que este es una modificación, donde se buscaba encontrar los usuarios que no consumieron un servicio determinado en el hotel. Por lo que primero se decidió encontrar los usuarios que, si consumieron, y seguido, comparar las identificaciones de los usuarios, los cuales no se encuentran dentro de los que si consumieron un servicio. Se tiene la misma cantidad anteriormente mencionada de registros en las tablas.

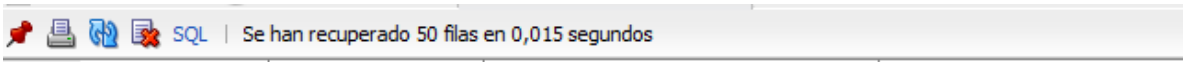
Este requerimiento funcional uso el índice anteriormente creado por el requerimiento funcional uno, para encontrar los rangos de fechas dados. Se sabe que el índice ocupa 503 bloques por lo que si son óptimos para su uso debido a que caben en memoria principal. Por lo que es un índice secundario disperso, usara un árbol b+ como estructura de datos, con apuntadores a memoria.

Como este requerimiento funcional es una extensión del anterior, se usó el mismo índice primario en la tabla apartan, donde se busca en un rango los servicios con un id pasado por parámetro.

Para verificar que los datos si concuerdan y el requerimiento funciona correctamente, se van a usar los mismos valores de prueba del requerimiento anterior, esto para observar que dichos valores no concuerdan, y que los valores presentados en el requerimiento anterior, no están presentes en este requerimiento.

2019010101	2019050101	3	0,032
2018050101	2019020101	5	0,04
2019030101	2019040101	9	0,022

Después de analizar el tiempo de consulta, se puede evidenciar que es más eficiente esta búsqueda que el requerimiento funcional anterior, sin importar si se ejecuta una operación extra para verificar los usuarios que no pertenecen al listado anterior. Su tiempo oscila entre 0,04 segundos y 0,01 segundos.



A continuación, se explica el plan de ejecución, y el porqué de que esta consulta es más eficiente que la anterior.

En el plan de consulta anterior se puede evidenciar que la opción del join varia a la del requerimiento funcional anterior. En un principio, el hash join entre la tabla sirven y el hash join de las tablas reserva y apartan, es de tipo right semi, lo cual significa que retorna la outter table, en la cual es la resultante de todos los registros que hicieron pareja en un inner join, al menos una vez. Esto sirve para que en el último hash join, el cual es de tipo anti, pueda obtener todas las identificaciones que no están dentro del right semi hash join. Por lo que retornara los registros que no entraron en el primer inner join. Como se puede observar, el select statement, va a retornar 3020 tuplas mientras que en requerimiento anterior va a retornar 5981 tuplas, por lo que su tiempo va a ser ligeramente mayor. Adicionalmente la suma de ambos resultados va a dar el total de tuplas de usuarios, por lo que se puede confirmar el éxito de la operación.

Plan de consulta propuesto

```
Select Statement
        Hash Join
                Usuario                                 Full Table Scan

                Hash Join
                        Reserva                         Full Table Scan

                        Nested Loop Join
                                Apartan                 Full Table Scan

                                Hash Join
                                        Sirven          Index Scan
                                                Filter Predicate on id=1, 2019010101<=FechaUso<=2019050101
                                        Habitacion      Unique Scan
```

El plan de consulta desarrollado por el grupo de trabajo es muy similar al planteado por Oracle. En un principio, se hace un index scan a la tabla sirven, ya que se debe encontrar las tuplas con ServicioId igual a 1. Por otro lado, toca leer todas las tuplas de habitación, para poder el join. Se realiza un Hash join, debido, a que toca juntarlos por una llave igual, esto genera que las colisiones sean correctas. Seguido se hace un Nested Loop Join entre la tabla resultante del hash anterior y la tabla apartan, la cual tiene que leerse completa en lectura secuencial rápida, cabe resaltar que al realizar el Nested Loop Join, significa que una de las tablas cabe en memoria. Después se hacen dos hashes joins, uno con la tabla reservas y otro con la tabla usuarios. A diferencia de la consulta anterior, se realiza un anti join, para determinar los usuarios que no pertenecen a dicha consulta, es decir al universo de usuarios, se le restan los usuarios pertenecientes a la consulta.

Requerimiento Funcional de Consulta 12

RFC12-Los buenos clientes son de tres tipos: aquellos que realizan estancias (las estancias están delimitadas por un check in y su respectivo check out) en HotelAndes al menos una vez por trimestre, aquellos que siempre consumen por lo menos un servicio costoso (Entiéndase como costoso, por ejemplo, con un precio mayor a $300.000.oo) y aquellos que en cada estancia consumen servicios de SPA o de salones de reuniones con duración mayor a 4 horas. Esta consulta retorna toda la información de dichos clientes, incluyendo aquella que justifica su calificación como buenos clientes. Esta operación es realizada únicamente por el gerente general de HotelAndes

Sentencia SQL

```sql
SELECT us.nombre,us.identificacion,serv.nombre
FROM USUARIO US
INNER JOIN RESERVA RES
ON US.IDENTIFICACION=RES.ID_USUARIO
INNER JOIN APARTAN AP
ON RES.ID=AP.IDRESERVA
INNER JOIN HABITACION HAB
ON AP.NUMEROHABITACION=HAB.NUMEROHABITACION
INNER JOIN SIRVEN SIRV
ON HAB.NUMEROHABITACION=SIRV.NUMEROHABITACION
INNER JOIN SERVICIOS SERV
ON SIRV.IDSERVICIO=SERV.ID
WHERE SERV.COSTO>=20
OR SERV.NOMBRE LIKE'%SPA%'
OR SERV.NOMBRE LIKE '%SALA REUNIONES%'
OR SERV.NOMBRE LIKE '%SALA CONFERENCIAS%'
OR res.fechainicio between 20180401 and 20180701
and res.fechainicio between 20180701 and 20180901
and res.fechainicio between 20180901 and 20190101
group by us.nombre,us.identificacion,serv.nombre
```

Para solucionar el requerimiento de consulta 12 se uso la anterior sentencia sql, la cual usaba las tablas usuario, reserva, apartan, habitación, sirven, servicios, juntándolas mediante joins para poder obtener el resultado deseado, en la siguiente sección, se van a mostrar los tamaños de las tablas usadas para solucionar el requerimiento funcional.

Este requerimiento funcional hace uso de dos índices primarios, el de reservas y el de usuarios para obtener los mismos. Debido a que dichos índices caben en memoria, su costo es de 0 por lo que volvió la consulta lo mas eficiente posible, adicionalmente gracias a esto, se puede usar el nested loop join, el cual es el mas eficiente teniendo una complejidad de la suma de la lectura secuencial rápida de las tablas usadas. A continuación, se muestra la información de ambos índices usados:

Usuario PK Índice

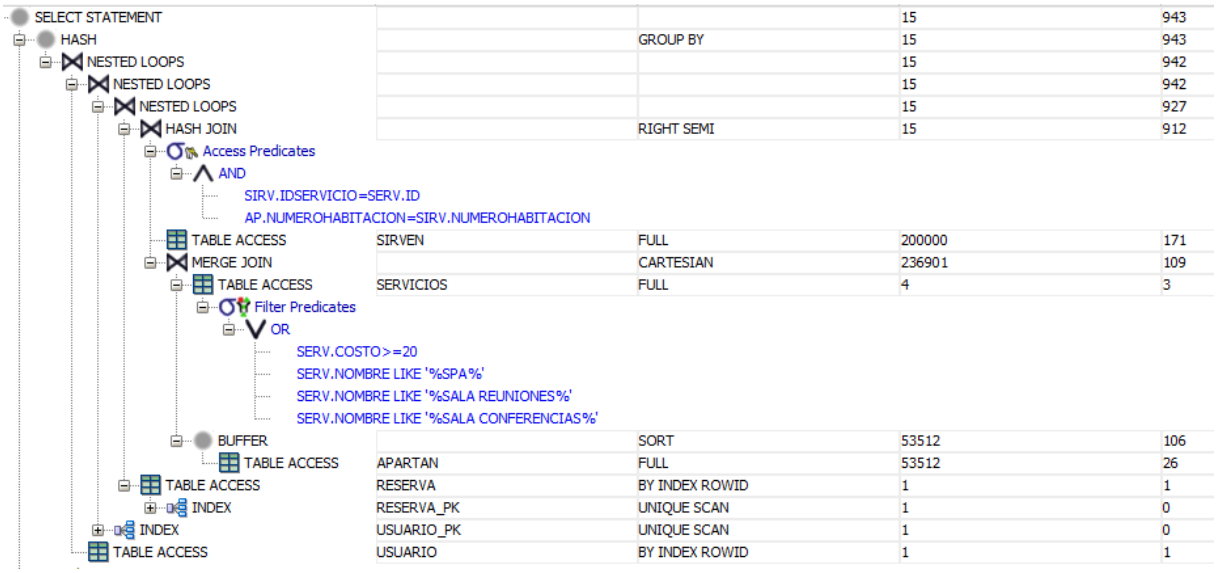
	Nombre	Valor
1	OWNER	ISIS2304A141910
2	INDEX_NAME	USUARIO_PK
3	TABLE_OWNER	ISIS2304A141910
4	TABLE_NAME	USUARIO
5	PARTITION_NAME	(null)
6	PARTITION_POSITION	(null)
7	SUBPARTITION_NAME	(null)
8	SUBPARTITION_POSITION	(null)
9	OBJECT_TYPE	INDEX
10	BLEVEL	1
11	LEAF_BLOCKS	17
12	DISTINCT_KEYS	9000
13	AVG_LEAF_BLOCKS_PER_KEY	1
14	AVG_DATA_BLOCKS_PER_KEY	1
15	CLUSTERING_FACTOR	68
16	NUM_ROWS	9000
17	AVG_CACHED_BLOCKS	(null)
18	AVG_CACHE_HIT_RATIO	(null)
19	SAMPLE_SIZE	9000
20	LAST_ANALYZED	18/05/19
21	GLOBAL_STATS	YES
22	USER_STATS	NO
23	STATTYPE_LOCKED	(null)
24	STALE_STATS	NO
25	SCOPE	SHARED

Reserva PK Índice

	Nombre	Valor
1	OWNER	ISIS2304A141910
2	INDEX_NAME	RESERVA_PK
3	TABLE_OWNER	ISIS2304A141910
4	TABLE_NAME	RESERVA
5	PARTITION_NAME	(null)
6	PARTITION_POSITION	(null)
7	SUBPARTITION_NAME	(null)
8	SUBPARTITION_POSITION	(null)
9	OBJECT_TYPE	INDEX
10	BLEVEL	1
11	LEAF_BLOCKS	18
12	DISTINCT_KEYS	10000
13	AVG_LEAF_BLOCKS_PER_KEY	1
14	AVG_DATA_BLOCKS_PER_KEY	1
15	CLUSTERING_FACTOR	60
16	NUM_ROWS	10000
17	AVG_CACHED_BLOCKS	(null)
18	AVG_CACHE_HIT_RATIO	(null)
19	SAMPLE_SIZE	10000
20	LAST_ANALYZED	13/05/19
21	GLOBAL_STATS	YES
22	USER_STATS	NO
23	STATTYPE_LOCKED	(null)
24	STALE_STATS	NO
25	SCOPE	SHARED

Al ser un índice primario y denso, la estructura de datos que se debería usar es una tabla de hash, para un rápido acceso al registro, mediante un UNIQUE SCAN, de esta manera se accede directamente al registro desde la llave. Esto sucede debido a que las tablas de hash en índices primarios contienen como llave el valor de la columna la cual se decidió el Primary key, y como hoja esta el registro completo. Cabe resaltar que el costo de obtener un registro mediante un índice primario es 0, ya que este esta ubicado directamente en la hoja, en memoria principal.

Plan de ejecución:



SELECT STATEMENT			15	943
HASH		GROUP BY	15	943
NESTED LOOPS			15	942
NESTED LOOPS			15	942
NESTED LOOPS			15	927
HASH JOIN		RIGHT SEMI	15	912
Access Predicates				
AND				
SIRV.IDSERVICIO=SERV.ID				
AP.NUMEROHABITACION=SIRV.NUMEROHABITACION				
TABLE ACCESS	SIRVEN	FULL	200000	171
MERGE JOIN		CARTESIAN	236901	109
TABLE ACCESS	SERVICIOS	FULL	4	3
Filter Predicates				
OR				
SERV.COSTO>=20				
SERV.NOMBRE LIKE '%SPA%'				
SERV.NOMBRE LIKE '%SALA REUNIONES%'				
SERV.NOMBRE LIKE '%SALA CONFERENCIAS%'				
BUFFER		SORT	53512	106
TABLE ACCESS	APARTAN	FULL	53512	26
TABLE ACCESS	RESERVA	BY INDEX ROWID	1	1
INDEX	RESERVA_PK	UNIQUE SCAN	1	0
INDEX	USUARIO_PK	UNIQUE SCAN	1	0
TABLE ACCESS	USUARIO	BY INDEX ROWID	1	1

En el plan de ejecución mostrado anteriormente se pueden evidenciar varios aspectos. En un principio, se filtran las condiciones de búsqueda sobre servicio, debido a que esta tabla solamente contiene 9 tuplas, por lo que su selección es rápida y no es costosa. Después de filtrar, se hace un merge join entre la tabla apartan y la tabla sirven. Después de la tabla resultante, se hace un hash join con la tabla sirven, debido a que es un join por un foreign key y las tablas no caben en memoria. Seguido se hace un nested loop jooin con la tabla reserva, como la tabla reserva cabe en memoria principal, se usan sus índices para hacer un nested loop join, el cual es el mas eficiente si alguna tabla cabe en memoria. El mismo proceso se lleva acabo con el join entre la tabla resultante anterior y la tabla usuario, debido a que los usuarios caben en memoria, se usan sus índices primarios, para realizar un nested loop join.

Plan de ejecucion

```
Select Statement
        Hash Join
                Usuario                                 Full Table Scan

                Nested Loop
                        Reserva                         Full Table Scan

                        Nested Loop Join
                                Apartan                 Full Table Scan

                                Nested loop
                                        Habitacion      Full table Scan

                                        Merge Join
                                                Sirven  Full Table Scan

                                                Servicio  Unique Scan
                                                        Filter predicate on costo>=20, nombre Like "Sala Conferencias"
```

En el plan de ejecucion planteado por el grupo de trabajo, se puede evidenciar un merge join, esto sucede por que al ser la selectividad muy baja de la tabla servicios, sobre los atributos en la condición del where, resulta mas efectivo que los demás joins. Por otro lado, se pueden evidenciar los nested loops predominando, esto debido a que almenos una de las tablas cabe en memoria, ya que según los bloques que usan, su tamaño es inferior al de la memoria principal. A continuación, se muestran los datos de cada una de las tablas.

Tablas

Apartan

	Nombre	Valor
1	NUM_ROWS	53512
2	BLOCKS	95
3	AVG_ROW_LEN	7
4	SAMPLE_SIZE	53512
5	LAST_ANALYZED	15/05/19
6	LAST_ANALYZED_SINCE	15/05/19

Consumen

	Nombre	Valor
1	NUM_ROWS	81057
2	BLOCKS	244
3	AVG_ROW_LEN	7
4	SAMPLE_SIZE	81057
5	LAST_ANALYZED	18/05/19
6	LAST_ANALYZED_SINCE	18/05/19

Convencion_usuario

	Nombre	Valor
1	NUM_ROWS	12745
2	BLOCKS	28
3	AVG_ROW_LEN	8
4	SAMPLE_SIZE	12745
5	LAST_ANALYZED	17/05/19
6	LAST_ANALYZED_SINCE	17/05/19

Convencion

	Nombre	Valor
1	NUM_ROWS	300
2	BLOCKS	5
3	AVG_ROW_LEN	107
4	SAMPLE_SIZE	300
5	LAST_ANALYZED	13/05/19
6	LAST_ANALYZED_SINCE	13/05/19

Habitacion

	Nombre	Valor
1	NUM_ROWS	100
2	BLOCKS	5
3	AVG_ROW_LEN	131
4	SAMPLE_SIZE	100
5	LAST_ANALYZED	13/05/19
6	LAST_ANALYZED_SINCE	13/05/19

Habitacion_convencion

	Nombre	Valor
1	NUM_ROWS	50000
2	BLOCKS	88
3	AVG_ROW_LEN	7
4	SAMPLE_SIZE	50000
5	LAST_ANALYZED	17/05/19
6	LAST_ANALYZED_SINCE	17/05/19

Habitacion_Mantenimientos

	Nombre	Valor
1	NUM_ROWS	100000
2	BLOCKS	370
3	AVG_ROW_LEN	16
4	SAMPLE_SIZE	100000
5	LAST_ANALYZED	17/05/19
6	LAST_ANALYZED_SINCE	17/05/19

Mantenimientos

	Nombre	Valor
1	NUM_ROWS	500
2	BLOCKS	13
3	AVG_ROW_LEN	125
4	SAMPLE_SIZE	500
5	LAST_ANALYZED	17/05/19
6	LAST_ANALYZED_SINCE	17/05/19

Ofrecen

	Nombre	Valor
1	NUM_ROWS	30000
2	BLOCKS	50
3	AVG_ROW_LEN	7
4	SAMPLE_SIZE	30000
5	LAST_ANALYZED	17/05/19
6	LAST_ANALYZED_SINCE	17/05/19

Planes

	Nombre	Valor
1	NUM_ROWS	4
2	BLOCKS	5
3	AVG_ROW_LEN	111
4	SAMPLE_SIZE	4
5	LAST_ANALYZED	13/05/19
6	LAST_ANALYZED_SINCE	13/05/19

Productos

	Nombre	Valor
1	NUM_ROWS	1000
2	BLOCKS	13
3	AVG_ROW_LEN	32
4	SAMPLE_SIZE	1000
5	LAST_ANALYZED	13/05/19
6	LAST_ANALYZED_SINCE	13/05/19

Reserva

	Nombre	Valor
1	NUM_ROWS	10000
2	BLOCKS	58
3	AVG_ROW_LEN	34
4	SAMPLE_SIZE	10000
5	LAST_ANALYZED	13/05/19
6	LAST_ANALYZED_SINCE	13/05/19

Reserva_Convencion

	Nombre	Valor
1	NUM_ROWS	5000
2	BLOCKS	28
3	AVG_ROW_LEN	27
4	SAMPLE_SIZE	5000
5	LAST_ANALYZED	19/05/19
6	LAST_ANALYZED_SINCE	19/05/19

Servicio_mantenimiento

	Nombre	Valor
1	NUM_ROWS	100000
2	BLOCKS	244
3	AVG_ROW_LEN	7
4	SAMPLE_SIZE	100000
5	LAST_ANALYZED	17/05/19
6	LAST_ANALYZED_SINCE	17/05/19

Servicios

	Nombre	Valor
1	NUM_ROWS	10
2	BLOCKS	5
3	AVG_ROW_LEN	63
4	SAMPLE_SIZE	10
5	LAST_ANALYZED	18/05/19
6	LAST_ANALYZED_SINCE	18/05/19

Servicio_convencion

	Nombre	Valor
1	NUM_ROWS	5000
2	BLOCKS	13
3	AVG_ROW_LEN	7
4	SAMPLE_SIZE	5000
5	LAST_ANALYZED	17/05/19
6	LAST_ANALYZED_SINCE	17/05/19

Sirven

	Nombre	Valor
1	NUM_ROWS	200000
2	BLOCKS	622
3	AVG_ROW_LEN	25
4	SAMPLE_SIZE	200000
5	LAST_ANALYZED	19/05/19
6	LAST_ANALYZED_SINCE	19/05/19

Usuarios

	Nombre	Valor
1	NUM_ROWS	9000
2	BLOCKS	73
3	AVG_ROW_LEN	73
4	SAMPLE_SIZE	9000
5	LAST_ANALYZED	19/05/19
6	LAST_ANALYZED_SINCE	19/05/19

Se llevaron a cabo diferentes procesos para la población de la base de datos. Para garantizar una coherencia en los datos, el primer mecanismo de generación de datos. Fue, generar los inserts manualmente, de la tabla servicios, planes y tipos de usuarios, debido a que estos deben tener unas características determinadas, para poder resolver los requerimientos de esta iteración. Por otra parte, se usó la plataforma mockaroo, para poblar las tablas que no necesitaban una gran cantidad de registros, tales como las convenciones y las habitaciones, ya que no se necesitaba una gran cantidad de estos para un correcto funcionamiento. El último proceso de la población de la base de datos fue la generación automática y aleatoria de los datos, mediante un programa creado en java. Dicho programa, generaba un archivo CSV con la información de cada tabla, la cual se va a importar a cada una de las tablas. A continuación, se ilustra el proceso de generación de los datos mediante la aplicación creada:

1) Se modifica el código del archivo para generar los datos deseados y el rango de estos
2) Se ejecuta la aplicación, se ingresa el nombre del archivo CSV y después se ingresa la cantidad de datos a generar.

```
Ingrese el nombre del archivo para cargar los datos:
TablaSirven
Se creo el archivo TablaSirven.csv en generatedData\TablaSirven.csv
Ahora ingrese el numero de datos que desea generar...
200
```

3) Se genera el archivo CSV con la cantidad de datos solicitada

301,1,1,63226,7086,231,Leone Jacobson,xavier@gmail.com
302,1,1,121063,114932,186,Werner Gleichner III,carmela@gmail.com
303,1,1,110440,62277,229,Westley Harvey Sr.,madisen@gmail.com
304,1,1,172580,129598,268,Fleta Weimann IV,myrl@gmail.com
305,1,1,42943,51901,53,Lessie Feest IV,rebeka@gmail.com
306,1,1,77336,64514,174,Sonny Kautzer,georgette@gmail.com
307,1,1,181614,148858,254,Bonita Cassin V,cathrine@gmail.com
308,1,1,99616,61747,243,Cielo Muller,kyra@gmail.com
309,1,1,41315,17047,138,Brendon Torp II,herminio@gmail.com
310,1,1,48638,138058,186,Mrs. Jordane Keeling,angus@gmail.com
311,1,1,177859,126097,51,Lance Block,lera@gmail.com
312,1,1,103171,168231,139,Chet Ward,carli@gmail.com
313,1,1,176647,17336,21,Dario Herman,judy@gmail.com
314,1,1,176350,100087,250,Jessie Kohler DVM,kaleb@gmail.com
315,1,1,75331,23244,272,Mr. Elvis Haag,virgie@gmail.com
316,1,1,18072,23964,160,Easton Padberg,pat@gmail.com
317,1,1,113505,136894,29,Vena Stracke,ezekiel@gmail.com
318,1,1,86301,151985,280,Ms. Hosea Kirlin,violet@gmail.com
319,1,1,198565,104994,205,Miss Marcia Homenick,dominique@gmail.com
320,1,1,107282,8167,280,Tre Hahn,coby@gmail.com
321,1,1,87440,61180,157,Makenna Franecki,marcus@gmail.com
322,1,1,95291,15584,132,Velma Pfannerstill,leland@gmail.com

4) Se debe importar el archivo CSV en la tabla deseada, para poblarla, Adicionalmente toca quitarle la cabecera al archivo para que pueda identificar bien las columnas.