PROYECTO DE CURSO – ITERACION 4

Juan Sebastián Sánchez, David Santiago Vargas
Universidad de los Andes, Bogotá, Colombia
{js.sanchezd1, ds.vargasp1}@uniandes.edu.co
Fecha de presentación: mayo 28 de 2023

Tabla de contenido

1. Introducción

2. Diseño de la aplicación

3. Selección de índices

4. Diseño físico, escenarios de prueba y análisis de eficiencia

5. Proceso de carga de datos

6. Análisis del proceso de optimización y el modelo de ejecución de consultas

7. Bibliografías

1. Introducción

El presente documento tiene como objetivo analizar el trabajo llevado a cabo para la elaboración de la iteración 4 del proyecto del caso AlohaAndes. En primer lugar, se presenta nuevas versiones para los modelos para satisfacer los nuevos requerimientos y restricciones. Posteriormente, se describen los tipos de índices seleccionados para cada requerimiento. Después se presentan los escenarios de prueba junto con un análisis de la eficiencia de cada requerimiento. Finalmente, se muestra cómo se llevó a cabo el proceso de carga de datos y se hacer un análisis en relación a la optimización de las consultas.

2. Diseño de la aplicación

En la figura 1 se muestra el modelo conceptual propuesto para la última iteración. Como los requerimientos funcionales a implementar son todos de consulta y no se introducen nuevos elementos dentro del mundo de negocio, se optó por mantener exactamente el mismo modelo UML de la iteración pasada. En otras palabras, no se agregó ninguna nueva clase ni tampoco ningún atributo nuevo a alguna ya existe

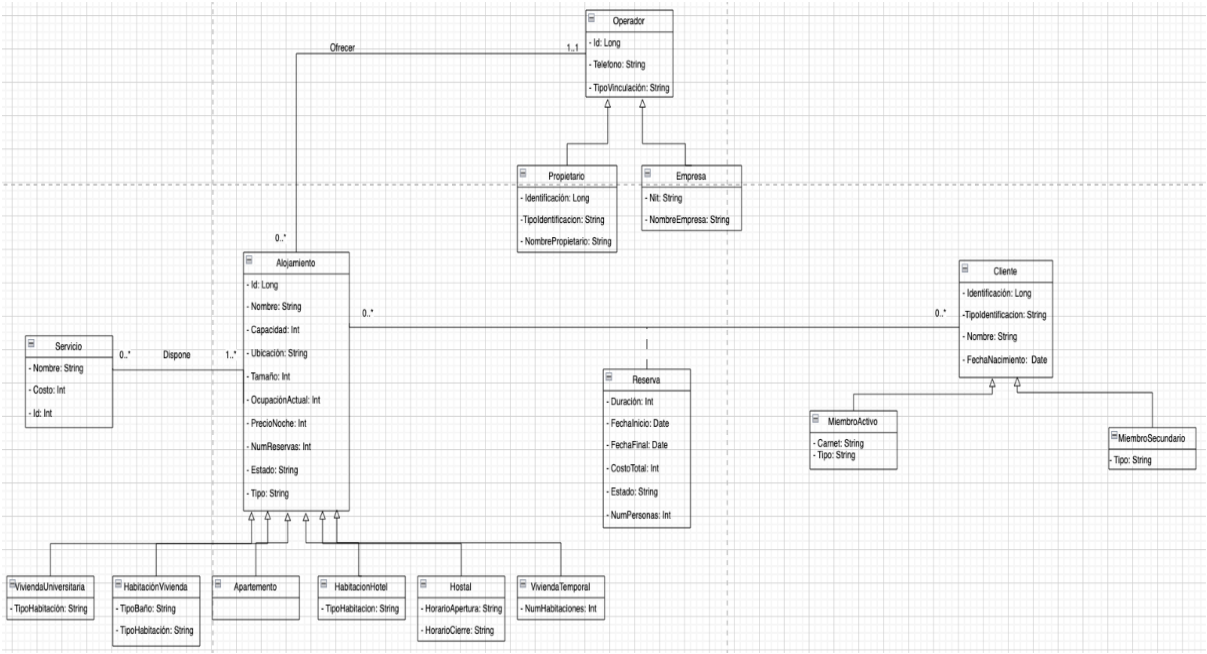


Figura 1: Diagrama UML con el modelo conceptual

Así como para el caso del modelo conceptual, el modelo relacional no sufrió ningún tipo de modificación respecto a aquella versión que se propuso para la iteración 3. A continuación, se muestran cada una de las relaciones del modelo relacional junto con sus correspondientes atributos y restricciones:

Operador

IdOperador	Teléfono	TipoVinculacion
PK, SA	NN, ND	NN
5018392	3172751312	Egresado
5018393	3192451234	Estudiante

Propietario

IdOperador	Identificación	TipoIdentificación	NombrePropietario
PK, FK (Operador.IdOperador)	NN, ND	NN, CK (‘CC’, ‘TI’, ‘PA’)	NN
5018393	1123489123	CC	Juan Felipe Gómez

Empresa

IdOperador	NIT	NombreEmpresa
PK, FK (Operador.IdOperador)	NN, ND	NN
5018392	739.830.012 - 3	Hilton

Alojamiento

IdAlojamiento	Nombre	Capacidad	Ubicación	Tamaño	PrecioNoche	OcupaciónActual	NumReservas	IdOperador	Estado	Tipo
PK, SA	NN	NN, CK (>0)	NN	NN, CK (>0)	NN, CK (>0)	NN, CK (>0)	NN, CK (>0)	NN, FK (Operador.IdOperador)	NN, CK (‘Habilitado’, ‘Deshabilitado’)	NN, CK1
1	HabitacionCityU	2	Centro	20	100000	0	50	5018393	Habilitado	ViviendaUniversitaria
2	Rosales	3	Sur	50	120000	2	70	5018393	Deshabilitado’	HabitacionVivienda
3	Las fuentes	4	Sur	100	80000	2	80	5018393	Habilitado	Apartamento
4	Habitación Hilton	2	Norte	20	200000	1	120	5018393	Habilitado	HabitacionHotel
5	El faro	15	Centro	70	45000	8	35	5018393	Deshabilitado’	Hostal
6	Apt 13 – Bella vista	1	Centro	70	65000	1	22	5018393	Habilitado	ViviendaTemporal

CK1: (‘ViviendaUniversitaria’, ‘HabitacionVivienda’,’ Apartamento’, ‘ HabitacionHotel’, ‘Hostal’, ‘ViviendaTemporal’)

ViviendaUniversitaria

IdAlojamiento	TipoHabitacion

PK, FK (Alojamiento.IdAlojamiento)	NN, CK (‘Compartida’, ‘Individual’)
1	Compartida

HabitacionVivienda

IdAlojamiento	TipoBaño	TipoHabitacion
PK, FK (Alojamiento.IdAlojamiento)	NN, CK (‘Privado’, ‘Publico’)	NN, CK (‘Compartida’, ‘Individual’)
2	Privado	Compartida

Apartamento

IdAlojamiento
PK, FK (Alojamiento.IdAlojamiento)
3

HabitacionHotel

IdAlojamiento	TipoHabitacion
PK, FK (Alojamiento.IdAlojamiento)	NN, CK (‘Estandar’, ‘Semisuites’, ‘Suites’)
4	

Hostal

IdAlojamiento	HorarioApertura	HorarioCierre
PK, FK (Alojamiento.IdAlojamiento)	NN	NN
5	18:00	10:00

ViviendaTemporal

IdAlojamiento	NumHabitaciones
PK, FK (Alojamiento.IdAlojamiento)	NN, CK (>0)
6	4

Cliente

IdCliente	TipoIdentificacion	NombreCliente	FechaNacimiento
PK, UA	NN, CK (‘CC’, ‘TI’, ‘PA’)	NN	NN
1000613198	CC	Santiago Vargas	09/12/2002
43724547	CC	Angela Prada	02/11/1969
1000658712	CC	Esteban Cuellar	04/01/2002
1447856981	CC	Juan Garavito	05/12/1989

MiembroActivo

IdMiembroActivo	Carnet	Tipo
PK, FK (Cliente.IdCliente)	NN, ND	NN, CK ('Estudiante', 'Profesor visitante', 'Empleado', 'Profesor titular')
1000613198	202013826	Estudiante
43724547	201914587	Empleado
1000658712	202014258	Estudiante

MiembroSecundario

IdMiembroSecundario	Tipo
PK, FK (Cliente.IdCliente)	NN, CK (‘Padre’, ‘Egresado’, ‘Invitado’)
1447856981	Padre

Servicio

IdServicio	Nombre	Costo
PK, SA	NN	NN, CK (>0)
101	Internet	50000
102	Cocineta	0

Dispone

IdServicio	IdAlojamiento
PK, FK (Servicio.IdServicio)	PK, FK (Alojamiento.IdAlojamiento)
101	1

Reserva

IdReserva	IdAlojamiento	IdCliente	Duracion	FechaInicio	Fecha Final	Costo Total	Estado	Num Personas	IdReservaColectiva
PK, SA	NN, FK (Alojamiento .IdAlojamiento)	NN, FK(Cliente.IdCliente)	NN, CK (>0)	NN	NN	NN, CK (>0)	NN, CK (‘Activa’, ‘Cancelada’,’Finalizada’)	NN, CK (>0)	

1	3	1000613198	15	15/01/2013	30/01/2013	1200000	Finalizada	2	NULL

3. Selección de índices

Para el desarrollo de los requerimientos funcionales de consulta propuestos para la presente entrega, tras deliberar y barajar las diferentes opciones de índices, se optó por no agregar ningún otro de los ya incluidos automáticamente por ORACLE al crear el esquema de la base de datos. Esto debido a que estos ya abarcan los datos de interes para las consultas y que los tiempos de búsqueda estan dentro de los límites establecidos para la iteración 4.

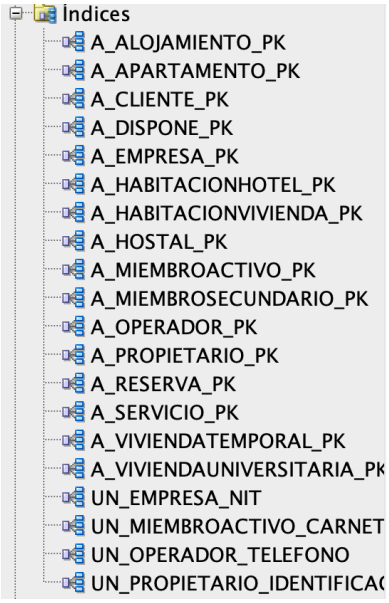


Figura 2: Índices por defecto creado por Oracle

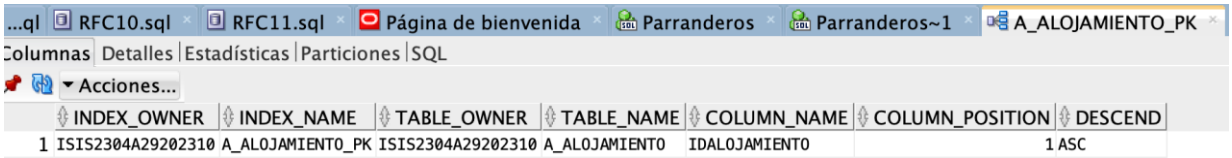


Figura 3: Ejemplo de información de uno de los índices creados por Oracle

4. Diseño físico, escenarios de prueba y análisis de eficiencia

4.1 RFC10

```sql
SELECT a_cliente.*
FROM a_cliente
INNER JOIN a_reserva ON a_cliente.idCliente = a_reserva.idCliente
WHERE ((A_Reserva.fechaInicio BETWEEN TO_DATE('2023-12-01', 'YYYY-MM-DD') AND
TO_DATE('2023-12-29', 'YYYY-MM-DD') ) AND (A_Reserva.fechaFinal BETWEEN
TO_DATE('2023-12-01', 'YYYY-MM-DD') AND TO_DATE('2023-12-29', 'YYYY-MM-DD') ))
GROUP BY a_cliente.idcliente, a_cliente.tipoidentificacion,
a_cliente.nombrecliente, a_cliente.fechanacimiento
ORDER BY a_cliente.idCliente
```

Al diseñar esta consulta, nuestro objetivo principal era maximizar la eficiencia y minimizar la sobrecarga del servidor de la base de datos. Para ello, utilizamos varias técnicas.
Primero, usamos el tipo de unión 'INNER JOIN', que es uno de los más eficientes en SQL. Este tipo de unión solo devuelve los registros que tienen una correspondencia en ambas tablas, lo que reduce significativamente el número de registros que el servidor necesita procesar.
En segundo lugar, utilizamos una cláusula WHERE para filtrar aún más los resultados. Nuestro filtro se basa en un rango de fechas específicas, lo que significa que el servidor no tiene que buscar en toda la base de datos, sino solo en un período de tiempo concreto. El uso de TO_DATE mejora aún más la eficiencia de esta operación, ya que convierte las cadenas de texto en fechas, lo que acelera la comparación.
Luego, aplicamos una cláusula GROUP BY, que agrupa los resultados por diferentes campos de la tabla 'a_cliente'. Esto nos permite reducir la cantidad de datos que se deben enviar al cliente, ya que cada grupo se representa como una única fila.
Por último, utilizamos la cláusula ORDER BY para ordenar los resultados por el campo 'idCliente'. Esto mejora la legibilidad de los resultados y facilita su interpretación.
En conjunto, todas estas técnicas contribuyen a la eficiencia de la consulta, ya que minimizan el número de registros que el servidor necesita procesar y el volumen de datos que se deben enviar al cliente.

Resultado de la Consulta

SQL | Se han recuperado 50 filas en 0,095 segundos

	IDCLIENTE	TIPOIDENTIFICACION	NOMBRECLIENTE	FECHANACIMIENTO
1	1	CC	Hannah Jenkins	11/10/06
2	61	TI	Charles Moss	11/12/94
3	62	TI	Paul Moss	15/04/30
4	71	PA	Chris Oconnell	19/03/42
5	86	CC	Laura Kane	05/06/19
6	90	TI	Amanda Copeland	06/04/63
7	100	CC	Dr. Amanda Pena	05/11/08
8	102	PA	James Lopez	04/01/71
9	138	CC	Christopher Hubbard	13/06/05
10	141	CC	Lynn Scott	29/06/08
11	144	CC	Lisa Marquez	05/06/35
12	151	CC	Latasha House	29/12/13
13	162	CC	Brian Davis	31/07/11
14	203	TI	Mary Edwards	22/09/90
15	213	TI	Jessica Smith	28/03/19
16	213	TI	Jessica Smith	28/03/19

Figura 4: Tiempo de ejecución del RFC10

Como se puede observar las estrategias utilizadas dieron los resultados esperados porque la consulta tardo tan solo 0.095 segundos, lo que es un tiempo verdaderamente pequeño teniendo en cuenta que son 1'000.000 de registros.

4.2 RFC11

```
SELECT a_cliente.*
FROM a_cliente
LEFT OUTER JOIN a_reserva ON a_cliente.idCliente = a_reserva.idCliente
WHERE a_cliente.idcliente NOT IN (SELECT a_cliente.idCliente
                                  FROM a_cliente
                                  INNER JOIN a_reserva ON
a_cliente.idCliente = a_reserva.idCliente
                                  WHERE a_reserva.idAlojamiento = 1 AND
((A_Reserva.fechaInicio BETWEEN TO_DATE('2023-12-01', 'YYYY-MM-DD') AND
TO_DATE('2023-12-29', 'YYYY-MM-DD') ) AND (A_Reserva.fechaFinal BETWEEN
TO_DATE('2023-12-01', 'YYYY-MM-DD') AND TO_DATE('2023-12-29', 'YYYY-MM-DD') ))
                                  GROUP BY a_cliente.idcliente)
GROUP BY a_cliente.idcliente, a_cliente.tipoidentificacion,
a_cliente.nombrecliente, a_cliente.fechanacimiento
ORDER BY a_cliente.idCliente;
```

Esta consulta fue diseñada con el objetivo de obtener un conjunto de clientes que no tienen una reserva para un alojamiento específico dentro de un rango de fechas establecido. Para lograr esto, empleamos varias técnicas de optimización.

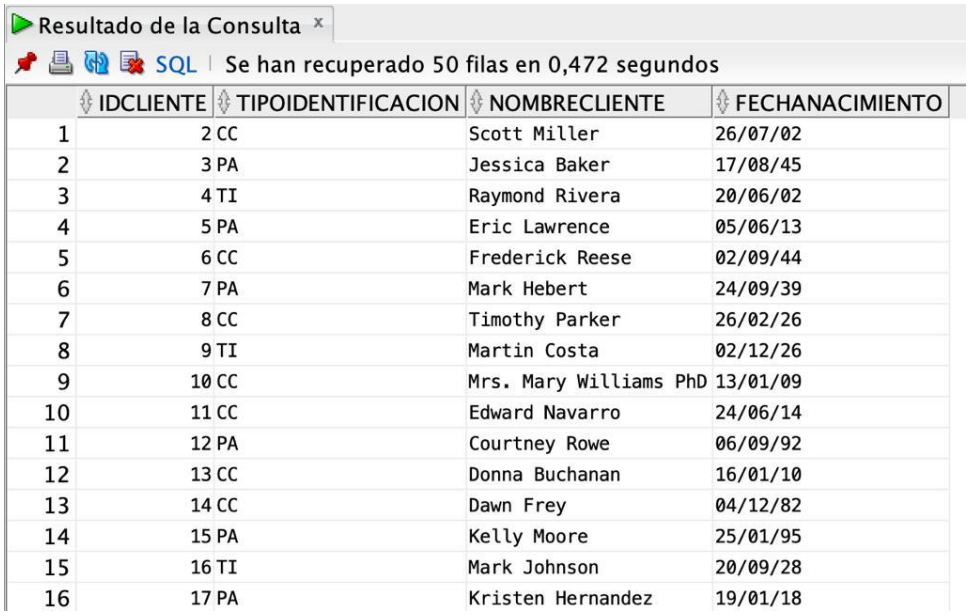
Para empezar, realizamos una 'LEFT OUTER JOIN' entre las tablas 'a_cliente' y 'a_reserva'. Esta operación es altamente eficiente ya que devuelve todos los registros de 'a_cliente' y los registros coincidentes de 'a_reserva'. Si no hay una coincidencia, el resultado es NULL del lado de 'a_reserva', lo cual ayuda a identificar aquellos clientes que no tienen reservas.

La cláusula WHERE con el operador 'NOT IN' se utiliza para filtrar los clientes que no se encuentran en la subconsulta especificada. Aunque las subconsultas pueden ser costosas en términos de rendimiento, en este caso, hemos implementado una subconsulta correlacionada que está optimizada al limitar la búsqueda a una condición específica ('a_reserva.idAlojamiento = 1') y a un rango de fechas definido.

El uso de TO_DATE en la subconsulta es igualmente eficaz para transformar cadenas en fechas, lo que acelera las comparaciones de fecha.

Además, la cláusula GROUP BY en la subconsulta y en la consulta principal ayuda a reducir la cantidad de datos redundantes, agrupando los resultados por distintos campos de 'a_cliente', lo que hace que el procesamiento posterior de los datos sea más manejable.

Por último, la cláusula ORDER BY ordena los resultados finales por 'idCliente', mejorando la presentación y legibilidad de los datos.


Resultado de la Consulta

SQL | Se han recuperado 50 filas en 0,472 segundos

	IDCLIENTE	TIPOIDENTIFICACION	NOMBRECLIENTE	FECHANACIMIENTO
1	2	CC	Scott Miller	26/07/02
2	3	PA	Jessica Baker	17/08/45
3	4	TI	Raymond Rivera	20/06/02
4	5	PA	Eric Lawrence	05/06/13
5	6	CC	Frederick Reese	02/09/44
6	7	PA	Mark Hebert	24/09/39
7	8	CC	Timothy Parker	26/02/26
8	9	TI	Martin Costa	02/12/26
9	10	CC	Mrs. Mary Williams PhD	13/01/09
10	11	CC	Edward Navarro	24/06/14
11	12	PA	Courtney Rowe	06/09/92
12	13	CC	Donna Buchanan	16/01/10
13	14	CC	Dawn Frey	04/12/82
14	15	PA	Kelly Moore	25/01/95
15	16	TI	Mark Johnson	20/09/28
16	17	PA	Kristen Hernandez	19/01/18

Figura 5: Tiempo de ejecución del RFC11

Como se puede observar las estrategias utilizadas dieron los resultados esperados porque la consulta tardo tan solo 0.472 segundos, lo que es un tiempo verdaderamente pequeño teniendo en cuenta que son 1'000.000 de registros.

4.3 RFC12

Para este requerimiento decidimos dividir la consulta en dos subconsultas diferentes, esto para que fuera más optimizado y que una no interfiriera en la otra.

```
--------Ocupación
    SELECT fechaInicial,
    MAX(CASE WHEN OCCUPATION = MAX_OCUPACION THEN idAlojamiento END) AS
ID_MAX_OCUPACION,
    MIN(CASE WHEN OCCUPATION = MIN_OCUPACION THEN idAlojamiento END) AS
ID_MIN_OCUPACION
FROM(
    SELECT a_reserva.idAlojamiento,
           TRUNC(a_reserva.fechaInicio, 'IW') fechaInicial,
           SUM(a_reserva.numPersonas) AS OCCUPATION,
           MAX(SUM(a_reserva.numPersonas)) OVER (PARTITION BY
TRUNC(a_reserva.fechaInicio, 'IW')) AS MAX_OCUPACION,
           MIN(SUM(a_reserva.numPersonas)) OVER (PARTITION BY
TRUNC(a_reserva.fechaInicio, 'IW')) AS MIN_OCUPACION
    FROM a_reserva
    GROUP BY a_reserva.idAlojamiento, TRUNC(a_reserva.fechaInicio, 'IW')
) subquery
GROUP BY fechaInicial
ORDER BY fechaInicial;
---------Operadores solicitados
SELECT fechaInicial,
```

```
    MAX(CASE WHEN SOLICITADOS = MAX_SOLICITADOS THEN idOperador END) AS
ID_MAX_SOLICITADOS,
    MIN(CASE WHEN SOLICITADOS = MIN_SOLICITADOS THEN idOperador END) AS
ID_MIN_SOLICITADOS
FROM(
    SELECT a_alojamiento.idOperador, TRUNC(a_reserva.fechaInicio, 'IW')
fechaInicial,
           COUNT(a_alojamiento.idOperador) AS SOLICITADOS,
           MAX(COUNT(a_alojamiento.idOperador)) OVER (PARTITION BY
TRUNC(a_reserva.fechaInicio, 'IW')) AS MAX_SOLICITADOS,
           MIN(COUNT(a_alojamiento.idOperador)) OVER (PARTITION BY
TRUNC(a_reserva.fechaInicio, 'IW')) AS MIN_SOLICITADOS
    FROM a_reserva
    INNER JOIN a_alojamiento ON a_alojamiento.idalojamiento =
a_reserva.idalojamiento
    GROUP BY a_alojamiento.idOperador, TRUNC(a_reserva.fechaInicio, 'IW')
) subquery
GROUP BY fechaInicial
ORDER BY fechaInicial;
```

Estas dos consultas están diseñadas para proporcionar una visión analítica de los datos de reserva y alojamiento, con un enfoque específico en la ocupación y la demanda de operadores durante la semana. Se han implementado técnicas de optimización para garantizar la eficiencia en el procesamiento y la precisión de los resultados.

En la primera consulta, se utiliza una subconsulta que agrupa las reservas por alojamiento y por semana (usando TRUNC con 'IW'). Este agrupamiento por semana permite un análisis más claro de la ocupación y la demanda a lo largo del tiempo. Dentro de la subconsulta, se emplean funciones de agregación (SUM, MAX y MIN) y la función de ventana OVER para calcular la ocupación máxima y mínima por semana. Las funciones de ventana son particularmente eficientes para este tipo de cálculos porque permiten realizar cálculos en un conjunto de filas relacionadas a una fila dada.

En la consulta principal, se utiliza MAX y MIN en combinación con CASE para seleccionar los alojamientos con la ocupación máxima y mínima para cada semana. Este enfoque asegura que solo se devuelva un alojamiento por cada semana, lo que facilita la interpretación de los resultados y minimiza la cantidad de datos que se deben enviar al cliente.

La segunda consulta es similar a la primera, pero se centra en los operadores de alojamiento en lugar de los alojamientos en sí. Se incorpora un JOIN adicional para conectar las tablas 'a_reserva' y 'a_alojamiento', lo que permite analizar la demanda de diferentes operadores de alojamiento. Al igual que en la primera consulta, se utilizan funciones de agregación y funciones de ventana para calcular la demanda máxima y mínima por semana, y luego se utilizan MAX, MIN y CASE para seleccionar los operadores con la demanda máxima y mínima para cada semana.

En resumen, estas consultas proporcionan un análisis eficiente y detallado de los datos de reserva y alojamiento, a pesar de la complejidad de los cálculos necesarios. La eficiencia se

logra a través de una combinación de funciones de agregación, funciones de ventana, operadores JOIN y la agrupación y filtrado cuidadoso de los datos.

Resultado de la Consulta | Resultado de la Consulta 1 | Res

SQL | Se han recuperado 50 filas en 0,193 segundos

	FECHAINICIAL	ID_MAX_OCUPACION	ID_MIN_OCUPACION
1	27/05/19	164089	7658
2	03/06/19	177979	414
3	10/06/19	177067	6489
4	17/06/19	178864	812
5	24/06/19	179788	4712
6	01/07/19	176063	5786
7	08/07/19	175687	4422
8	15/07/19	107040	912
9	22/07/19	179024	6650
10	29/07/19	173620	537
11	05/08/19	171753	2871
12	12/08/19	171538	843
13	19/08/19	178082	1419
14	26/08/19	179864	10039
15	02/09/19	68471	3479
16	09/09/19	168794	1986

Figura 6: Tiempo de ejecución del RFC12 – Parte A

Resultado de la Consulta | Resultado de la Consulta 1 | Resul

SQL | Se han recuperado 50 filas en 0,416 segundos

	FECHAINICIAL	ID_MAX_SOLICITADOS	ID_MIN_SOLICITADOS
1	27/05/19	195292	253
2	03/06/19	199983	374
3	10/06/19	90110	72
4	17/06/19	12195	1023
5	24/06/19	199390	298
6	01/07/19	199760	199
7	08/07/19	199260	172
8	15/07/19	73143	102
9	22/07/19	199972	135
10	29/07/19	199951	660
11	05/08/19	199507	350
12	12/08/19	199594	160
13	19/08/19	58608	267
14	26/08/19	151684	641
15	02/09/19	26571	1038
16	09/09/19	199660	47

Figura 7: Tiempo de ejecución del RFC12 – Parte B

Se puede observar que la estrategia de dividir esta consulta en dos funcionó adecuadamente poque al sumar el tiempo de cada subconsulta, se obtiene que el tiempo de ejecución total es de 0.609 segundos así que funcionó de manera muy optima teniendo en cuenta que son 1'000.000 de registros.

4.4 RFC13

```
SELECT DISTINCT a_cliente.*
FROM a_cliente
JOIN a_reserva ON a_cliente.idCliente = a_reserva.idCliente
JOIN a_alojamiento ON a_reserva.idAlojamiento = a_alojamiento.idAlojamiento
WHERE
    -- Clientes que hacen reservas al menos una vez al mes
    EXISTS (
        SELECT 1
        FROM a_reserva
        WHERE a_cliente.idCliente = a_reserva.idCliente
        GROUP BY TRUNC(a_reserva.fechaInicio, 'MM')
        HAVING COUNT(*) >= 1
    )
    OR
    -- Clientes que siempre reservan alojamientos costosos
    EXISTS (
        SELECT 1
        FROM a_reserva
        INNER JOIN a_alojamiento ON a_reserva.idAlojamiento =
a_alojamiento.idAlojamiento
        WHERE a_cliente.idCliente = a_reserva.idCliente AND
a_alojamiento.precioNoche >= 650000
    )
    OR
    -- Clientes que siempre reservan suites
    EXISTS (
        SELECT 1
        FROM a_reserva
        INNER JOIN a_alojamiento ON a_reserva.idAlojamiento =
a_alojamiento.idAlojamiento
        INNER JOIN a_habitacionHotel ON a_habitacionHotel.idAlojamiento =
a_alojamiento.idAlojamiento
        WHERE a_cliente.idCliente = a_reserva.idCliente AND
a_habitacionHotel.tipoHabitacion = 'Suites'
    );
```

Esta consulta fue diseñada para identificar un conjunto de clientes basado en tres criterios de comportamiento de reserva. Para garantizar la eficiencia de la consulta, se emplearon varias estrategias de optimización y diseño de consultas SQL.

En primer lugar, se utilizan tres operadores JOIN para conectar las tablas relevantes ('a_cliente', 'a_reserva', 'a_alojamiento' y 'a_habitacionHotel'). Los operadores JOIN son eficientes en SQL y ayudan a reducir la cantidad de datos que necesitan ser procesados al limitar los resultados a las coincidencias entre las tablas.

Para cada uno de los criterios de comportamiento de reserva, se utiliza la cláusula EXISTS, que se detiene en cuanto encuentra un registro que cumple con la condición, lo que la hace más eficiente que otras cláusulas como IN, especialmente cuando se espera que exista una coincidencia.

El primer EXISTS se utiliza para identificar los clientes que realizan al menos una reserva al mes. Se aplica la función TRUNC a 'fechaInicio' para agrupar por mes, y luego se usa la cláusula HAVING para filtrar aquellos que tienen una cuenta de al menos 1, lo que significa que han realizado al menos una reserva en ese mes.

El segundo y tercer EXISTS se utilizan para identificar clientes que siempre reservan alojamientos costosos y suites respectivamente. Ambos utilizan un INNER JOIN para conectar las tablas relevantes y luego filtran los resultados basándose en el precio por noche o el tipo de habitación.

Además, la consulta utiliza SELECT DISTINCT para eliminar cualquier duplicado que pueda surgir al unirse a las otras tablas, lo que garantiza que cada cliente se represente solo una vez en los resultados.

Por último, la consulta utiliza la cláusula OR para combinar estos criterios, lo que significa que un cliente que cumpla cualquiera de estos criterios será incluido en los resultados. Esto es más eficiente que ejecutar tres consultas separadas y luego combinar los resultados.

En resumen, la eficiencia de esta consulta se debe a la combinación de operadores JOIN eficientes, el uso de EXISTS para una búsqueda rápida, la agrupación y filtrado eficiente de los datos, y el uso de SELECT DISTINCT para eliminar duplicados.

SQL | Se han recuperado 50 filas en 0,193 segundos

	IDCLIENTE	TIPOIDENTIFICACION	NOMBRECLIENTE	FECHANACIMIENTO
1	29	TI	Wanda Lopez	06/10/84
2	64	CC	Levi Collins	10/01/37
3	76	PA	Kevin Clarke	07/05/80
4	81	CC	Lauren Burke	12/10/31
5	86	CC	Laura Kane	05/06/19
6	89	PA	Katherine Young	15/12/67
7	145	TI	Elaine Flores	24/03/28
8	199	TI	Nathaniel Brewer	08/09/19
9	201	CC	Pamela Johnson	27/03/60
10	205	CC	Angel Mcconnell	04/10/26
11	216	TI	Deanna Bowman	14/11/36
12	226	CC	John Simpson	10/02/13
13	254	TI	Randall Smith	06/10/76
14	255	CC	Miss Jennifer Baker	13/08/52
15	290	PA	Cassandra Hughes	14/02/53
16	302	TI	Sean Thomas	27/07/80
17	308	PA	Kim Copeland	02/04/31

Figura 8: Tiempo de ejecución del RFC13

Como se puede observar las estrategias utilizadas dieron los resultados esperados porque la consulta tardo tan solo 0.193 segundos, lo que es un tiempo verdaderamente pequeño, teniendo en cuenta que son 1'000.000 de registros.

5. Proceso de carga de datos

Con respecto a la carga de datos, debido al alto volumen de información a insertar para la presente iteración, resultaba prácticamente inviable crear un script SQL con las todas las inserciones de tuplas a realizar. Por lo tanto, se optó por utilizar la función de importar nuevos datos para una tabla a partir de un archivo CSV.

Para la generación automatizada de archivos CSV se utilizaron las librerías de Python denominadas ‘fake’ y ‘pandas’. Esta primera permite generar nombres, fechas, correos, números en determinado rango, entre otros. Por su parte, ‘pandas’ se usa para construir una estructura ‘dateframe’ para cada tabla, de manera que puedan ser posteriormente utilizadas para crear los archivos de formato CSV. A continuación, se muestra un fragmento del código creado para la carga de datos:

```
# CSV OPERADOR
for op in range(1,200000):
    fake_data_operador["idOperador"].append( op )
    operador_index.append(op)
    fake_data_operador["telefono"].append( fake.unique.random_int(3000000000, 3999999999) )
    fake_data_operador["tipoVinculacion"].append( fake.tipo_vinculacion() )


df_fake_data_operador = pd.DataFrame(fake_data_operador)
df_fake_data_operador.to_csv("csv loader/operadores.csv", index=False)

#Provider para operadores
operador_index_provider = DynamicProvider(
    provider_name="idOperador",
    elements=operador_index
)

# CSV PROPIETARIO
for pro in range(1,100001):
    fake_data_propietario["idOperador"].append( pro )
    operador_index.append(pro)
    fake_data_propietario["identificacion"].append( fake.unique.random_int(1000000000, 1999999999) )
    fake_data_propietario["tipoIdentificacion"].append( fake.tipo_identificacion )
    fake_data_propietario["nombrePropietario"].append( fake.name() )

df_fake_data_propietario = pd.DataFrame(fake_data_propietario)
df_fake_data_propietario.to_csv("csv loader/propietarios.csv", index=False)

# CSV EMPRESA
for emp in range(100001,200001):
    fake_data_empresa["idOperador"].append( emp )
    operador_index.append(emp)
    fake_data_empresa["nit"].append( fake.unique.random_int(1000000000, 1999999999) )
    fake_data_empresa["nombreEmpresa"].append( fake.name() )

df_fake_data_empresa = pd.DataFrame(fake_data_empresa)
df_fake_data_empresa.to_csv("csv loader/empresas.csv", index=False)
```

Figura : Fragmento del código para crear archivos CSV (Docs/sqlLoader.py)

Tras haber construido los archivos CSV necesarios, para cada tabla se seleccionó la opción para importar datos y se escogió el archivo respectivo. Finalmente, para cada atributo se especificó la columna del archivo importado y se aplicaron algunas configuraciones menores para persistir las nuevas tuplas en la base de datos.

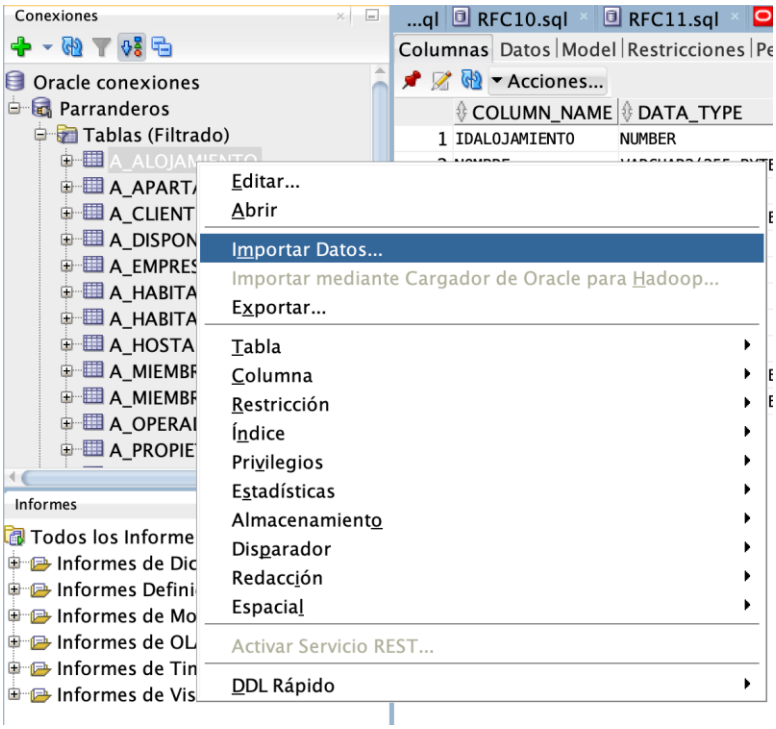


Figura 10: Opción para importar nuevos datos por tabla en SQL developer

6. Análisis del proceso de optimización y el modelo de ejecución de consultas

6.1 RFC10

A continuación, se muestra el plan de ejecución del RFC10, el cual se comprueba que la manera en que se planteó permite que sea muy eficaz, además que los índices que nos brinda el propio SQL Oracle son lo suficientemente óptimos para cumplir con el RFN6 que es que cada consulta se demore menos de 0.8 segundos.

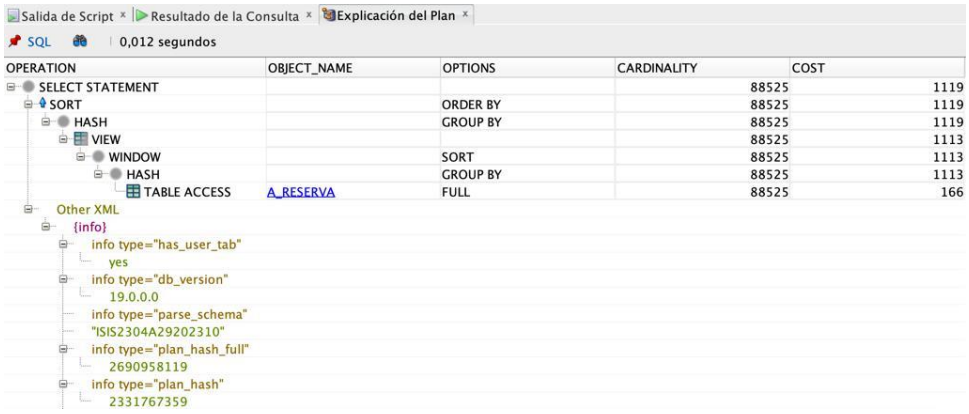


Figura 11: Plan de ejecución de RFC10

6.1 RFC11

A continuación, se muestra el plan de ejecución del RFC11, el cual se comprueba que la manera en que se planteó permite que sea muy eficaz, además que los índices que nos brinda el propio SQL Oracle son lo suficientemente óptimos para cumplir con el RFN6 que es que cada consulta se demore menos de 0.8 segundos.

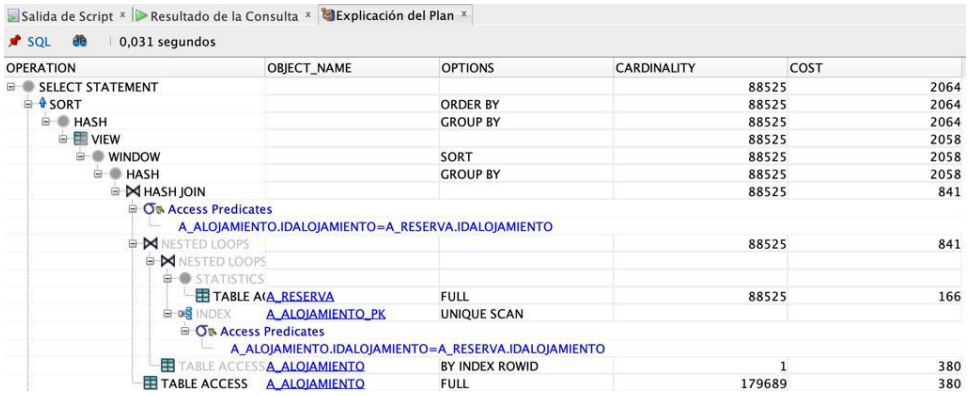


Figura 12: Plan de ejecución de RFC11

6.1 RFC12

A continuación, se muestra el plan de ejecución del RFC12 Parte A, el cual se comprueba que la manera en que se planteó permite que sea muy eficaz, además que los índices que nos brinda el propio SQL Oracle son lo suficientemente óptimos para cumplir con el RFN6 que es que cada consulta se demore menos de 0.8 segundos.

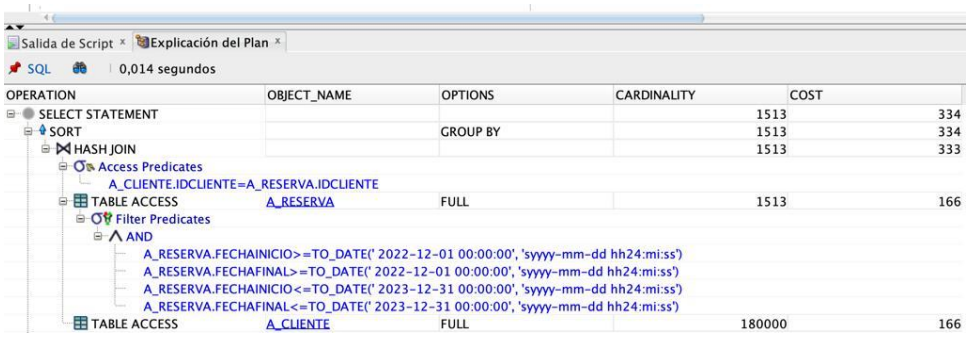


Figura 13: Plan de ejecución de RFC12- Parte A

A continuación, se muestra el plan de ejecución del RFC12 Parte B, el cual se comprueba que la manera en que se planteó permite que sea muy eficaz, además que los índices que nos brinda el propio SQL Oracle son lo suficientemente óptimos para cumplir con el RFN6 que es que cada consulta se demore menos de 0.8 segundos.

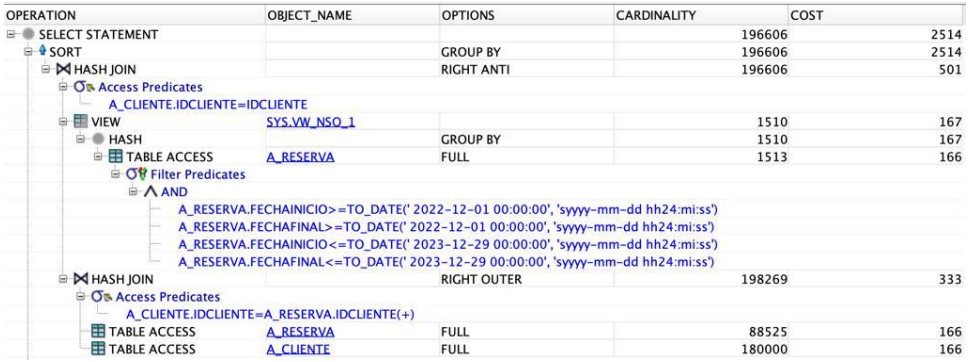
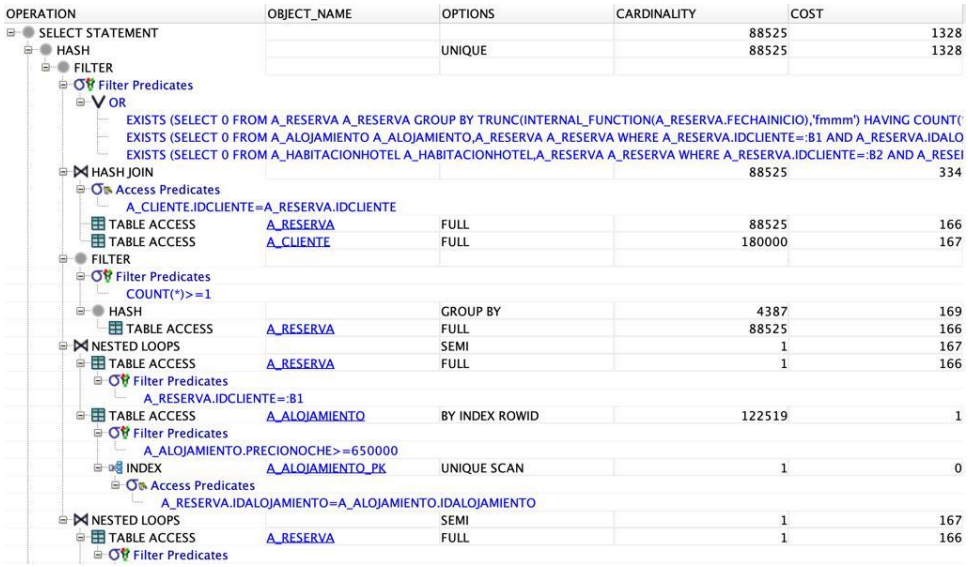


Figura 13: Plan de ejecución de RFC12- Parte B

6.1 RFC13

A continuación, se muestra el plan de ejecución del RFC13, el cual se comprueba que la manera en que se planteó permite que sea muy eficaz, además que los índices que nos brinda el propio SQL Oracle son lo suficientemente óptimos para cumplir con el RFN6 que es que cada consulta se demore menos de 0.8 segundos



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			88525	1328
HASH		UNIQUE	88525	1328
FILTER				
Filter Predicates				
OR				
EXISTS (SELECT 0 FROM A_RESERVA A_RESERVA GROUP BY TRUNC(INTERNAL_FUNCTION(A_RESERVA.FECHAINICIO),'fmmm') HAVING COUNT(				
EXISTS (SELECT 0 FROM A_ALOJAMIENTO A_ALOJAMIENTO,A_RESERVA A_RESERVA WHERE A_RESERVA.IDCLIENTE=:B1 AND A_RESERVA.IDALO				
EXISTS (SELECT 0 FROM A_HABITACIONHOTEL A_HABITACIONHOTEL,A_RESERVA A_RESERVA WHERE A_RESERVA.IDCLIENTE=:B2 AND A_RESEI				
HASH JOIN			88525	334
Access Predicates				
A_CLIENTE.IDCLIENTE=A_RESERVA.IDCLIENTE				
TABLE ACCESS	A_RESERVA	FULL	88525	166
TABLE ACCESS	A_CLIENTE	FULL	180000	167
FILTER				
Filter Predicates				
COUNT(*)>=1				
HASH		GROUP BY	4387	169
TABLE ACCESS	A_RESERVA	FULL	88525	166
NESTED LOOPS		SEMI	1	167
TABLE ACCESS	A_RESERVA	FULL	1	166
Filter Predicates				
A_RESERVA.IDCLIENTE=:B1				
TABLE ACCESS	A_ALOJAMIENTO	BY INDEX ROWID	122519	1
Filter Predicates				
A_ALOJAMIENTO.PRECIONOCHE>=650000				
INDEX	A_ALOJAMIENTO_PK	UNIQUE SCAN	1	0
Access Predicates				
A_RESERVA.IDALOJAMIENTO=A_ALOJAMIENTO.IDALOJAMIENTO				
NESTED LOOPS		SEMI	1	167
TABLE ACCESS	A_RESERVA	FULL	1	166
Filter Predicates				

Figura 14: Plan de ejecución de RFC13

7. Bibliografías

[1] SILBERSCHATZ, Abraham, KORTH, Henry F., SUDARSHAN, S. “D*atabase system concepts*”. Seventh edition. New York, NY: McGraw-Hill, [2020].

[2] LEWIS, Philip, BERNSTEIN, Arthur, KIFER, Michael. "*Database Systems- An ApplicationOriented Approach*". Second Edition. Addison-Wesley, 2006.

[3] GARCIA-MOLINA, Hector, ULLMAN, Jeffrey, WIDOM, Jennifer. "*Database Systems: The complete Book*". 2nd. Edition. Prentice Hall, 2009.

[4] BAGUI, S., & EARP, R. (2003). Database Design Using Entity-Relationship Diagrams. Auerbach Publications, Boca Raton, Florida: CRC Press.