“Iteración 4: Alohandes”

José D. Flórez Ruiz, Carlos M. Muñoz Almeida
Iteración 4 - Documento
Universidad de los Andes, Bogotá, Colombia
{jd.florezr1, c.munoza}@uniandes.edu.co
Fecha de presentación: Mayo 28 de 2023

Contenido

1. Introducción

En el presente documento, se revisará el impacto de nuevos requerimientos de la aplicación de Alohandes un sistema dedicado a la publicación y reserva de alojamientos para miembros de una comunidad universitaria. También, se realizará un análisis enfocado en la eficiencia de los requerimientos de consulta ofrecidos en Alohandes y se detallara que cambios en el esquema físico de la base de datos fueron necesarios para lograr garantizar la eficiencia en las consultas y en actualizaciones.

2. Diseño de la aplicación

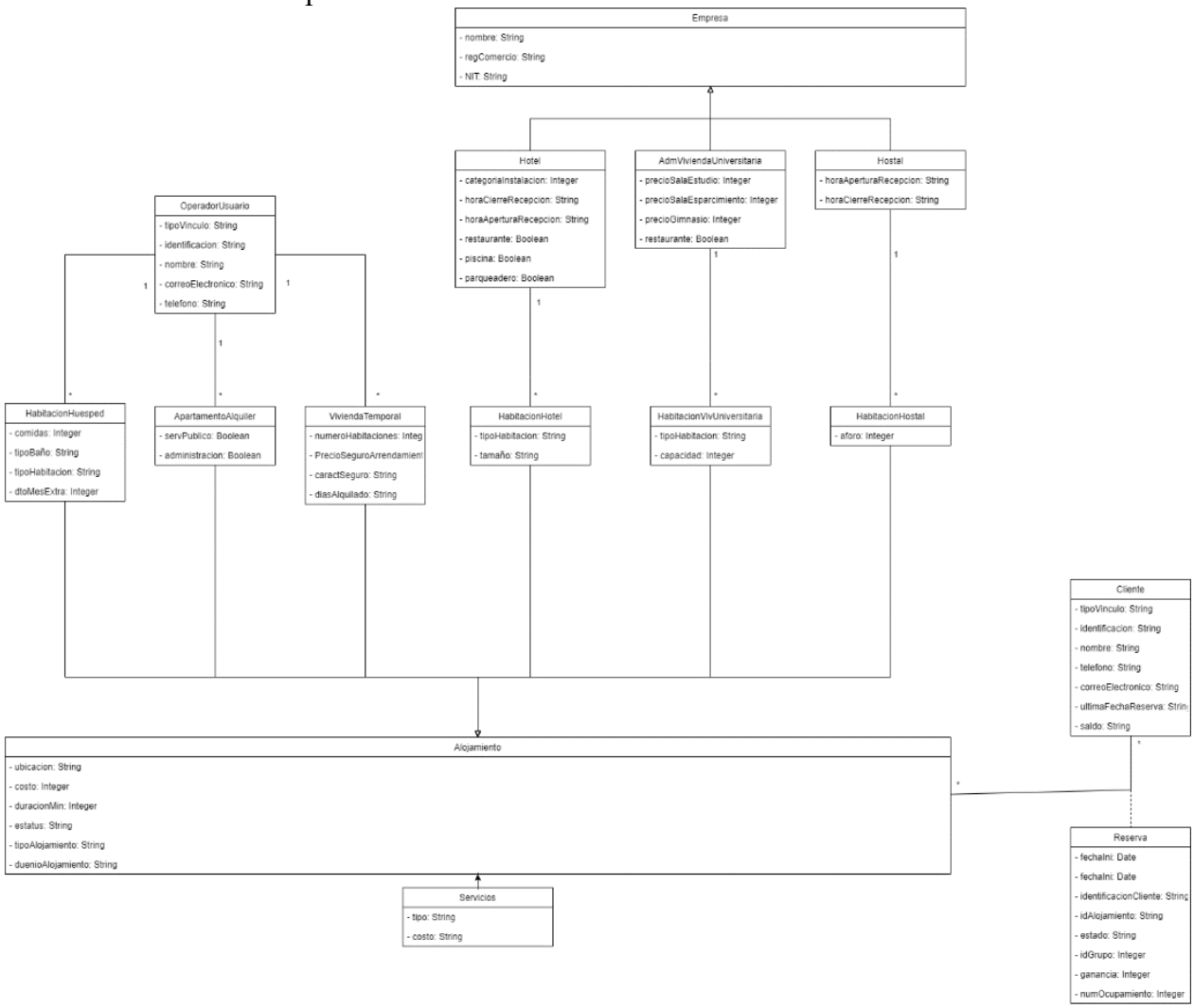
2.1. Impacto de los nuevos requerimientos

A gran escala podemos decir que estos fueron los principales cambios realizados, en secciones posteriores se detallara mucho más a fondo cual fue la utilidad de estos cambios y como se realizó la gestión de eficiencia para cada requerimiento de consulta. Para la correcta implementación de los nuevos requerimientos se tuvieron que realizar los siguientes cambios enfocados en garantizar:

- En la tabla Reserva se agregaron los siguientes atributos:
  - NumOcupamiento: Muestra la cantidad de personas que van a ocupar el alojamiento reservado.
- Nuevos atributos en Alojamiento:
  - TipoAlojamiento: Muestra el tipo de alojamiento.
  - IdOperador: Indica el operador al cual le corresponde ese alojamiento.
- Creación de índices:
  - En la tabla Reserva un índice dado por Fecha_Ini, y otro dado por Fecha_Ini y IdAlojamiento
  - En la tabla Alojamiento un índice dado por Id, y IdOperador

(La justificación sobre la creación de estos índices y su respectivo uso se menciona mucho más a fondo en secciones posteriores del documento)

2.2. Modelo conceptual



2.3. Modelo relacional

Relación

At 1	At 2	At 3	At 4

OperadorUsuario

identificacion	nombre	tipoVinculo	correoElectronico	telefono
PK, UA	NN	NN, CK5	NN	NN

HabitacionHuesped

idAlojamiento	comidas	tipoBaño	tipoHabitacion	dtoMesExtra	IDENTIFICACIONOPERADORUSUARIO
PK, FK Alojamiento.id	NN, CK >= 0	NN, CK2	NN, CK3	NN, 100 >= CK > 0	NN, FK OperadorUsuario.identificacion

ApartamentoAlquiler

idAlojamiento	servPublico	administracion	IDENTIFICACIONOPERADORUSUARIO
PK, FKAlojamiento.id	NN, CK1	NN, CK1	NN, FKOperadorUsuario.identificacion

Restricciones : El apartamento solo podra ser alquilado por un propietario que tenga algun tipo de vinculo con la comunidad uniandina.

ViviendaTemporal

idAlojamiento	numeroHabitaciones	PRECIOSEGUROARRENDAMIENTO	cancSeguro	diasAlquilado	propietario
PK, FKAlojamiento.id	NN, CK > 0	NN, CK > 0	NN	NN, CK <= 30	NN, FKOperadorUsuario.identificacion

Cliente

identificacion	nombre	tipoVinculo	correoElectronico	telefono	ultimaFechaReserva	Saldo
PK, UA	NN	NN, CK5	NN	NN	NN	NN

Reserva

id	fechaIni	fechaFin	identificacionCliente	idAlojamiento	idAlojamiento	Estado	IdGrupo	Ganancia	numOcupamiento
PK, SA	NN	NN	NN, FKCliente.identificacion	NN, FKReserva.id	NN, FKReserva.id	NN, CK1	NN	NN	NN

Alojamiento

id	ubicacion	duracionMin	Costo	Estatus	TipoAlojamiento	duenioAlojamiento
PK, SA	NN,	NN, CK > 0	NN, CK > 0	NN, CK1	NN, CK8	NN, CK8

Hotel

regComercio	NIT	nombre	restaurante	parqueadero	piscina
PK, UA	NN	NN	NN, CK1	NN, CK1	NN, CK1

HabitaciónHotel

id	tipoHabitación	tamaño	idHotel
PK, FKAlojamiento.id	NN, CK6	NN	**NN, FKHotel.regComercio**

ViviendaUniversitaria

regComercio	NIT	nombre	precioSalaEstudio	precioSalaEsparcimiento	precioGimnasio	Restaurante
PK, UA	NN	NN	NN, CK(>=0)	NN, CK(>=0)	NN, CK(>=0)	NN

HabitaciónVivUniversitaria

id	tipoHabitación	capacidad	idViviendaUniversitaria
PK, FKAlojamiento.id	NN	CK(>0)	**NN, FKAdmViviendaUniversitaria.regComercio**

Hostal

regComercio	NIT	nombre	horaAperturaRecepción	horaCierreRecepción
PK, UA	NN	NN	NN	NN

HabitaciónHostal

id	aforo	idhostal
PK, FKAlojamiento.id	CK(>=0)	**NN, FKHostal.regComercio**

Servicio

id	tipo
PK	NN,CK 7

AlojamientoServicio

idAlojamiento	idServicio	Costo
PK, FKAlojamiento.id	FKServicio.id	CK (>=0)

3. Diseño Físico

3.1. Distribución de los datos

A continuación, se presenta como se distribuyeron alrededor de un millón de datos en todo nuestro sistema:

Distribución de datos	
Tabla	Porcentaje(%)
OperadorUsuario	5
HabitaciónHuesped	5
ApartamentoAlquiler	5
ViviendaTemportal	5
Hotel	2
AdmViviendaUniversitaria	2
Hostal	2
HabitaciónHotel	4
HabitaciónVivUniversitaria	4
HabitaciónHostal	4
Alojamiento	27
AlojamientoServicio	10
Cliente	10
Reserva	15
Total	100

Por medio de esta distribución se realizaron las pruebas y se obtuvieron los tiempos que se presentaran para cada requerimiento funcional de consulta.

3.2. Índices

El uso de índices fue de vital importancia para garantizar el rendimiento y eficiencia de la aplicación.

```
SELECT *
FROM ALL_INDEXES
WHERE OWNER = 'ISIS2304C19202310';
```

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	
1	ISIS2304C19202310	IDX_ALOJAMIENTO_ID_IDOPERADOR	NORMAL	ISIS2304C19202310	A_ALOJAMIENTO	TA
2	ISIS2304C19202310	PK_HOTEL	NORMAL	ISIS2304C19202310	A_HOTEL	TA
3	ISIS2304C19202310	PK_HABITACIONHOTEL	NORMAL	ISIS2304C19202310	A_HABITACIONHOTEL	TA
4	ISIS2304C19202310	PK_VIVIENDAUNIVERSITARIA	NORMAL	ISIS2304C19202310	A_VIVIENDAUNIVERSITARIA	TA
5	ISIS2304C19202310	A_CLIENTE_PK	NORMAL	ISIS2304C19202310	A_CLIENTE	TA
6	ISIS2304C19202310	PK_HABITACIONVIVIENDAUNIVERSITARIA	NORMAL	ISIS2304C19202310	A_HABITACIONVIVIENDAUNIVERSITARIA	TA
7	ISIS2304C19202310	PK_HOSTAL	NORMAL	ISIS2304C19202310	A_HOSTAL	TA
8	ISIS2304C19202310	A_OPERADORUSUARIO_PK	NORMAL	ISIS2304C19202310	A_OPERADORUSUARIO	TA
9	ISIS2304C19202310	IDX_RESERVA_FECHAINI	FUNCTION-BASED NORMAL	ISIS2304C19202310	A_RESERVA	TA
10	ISIS2304C19202310	PK_SERVICIO	NORMAL	ISIS2304C19202310	A_SERVICIO	TA
11	ISIS2304C19202310	PK_EMPRESASERVICIO	NORMAL	ISIS2304C19202310	A_ALOJAMIENTOSERVICIO	TA
12	ISIS2304C19202310	A_ALOJAMIENTO_PK	NORMAL	ISIS2304C19202310	A_ALOJAMIENTO	TA
13	ISIS2304C19202310	A_VIVIENDATEMPORAL_PK	NORMAL	ISIS2304C19202310	A_VIVIENDATEMPORAL	TA
14	ISIS2304C19202310	A_RESERVA_PK	NORMAL	ISIS2304C19202310	A_RESERVA	TA
15	ISIS2304C19202310	PK_HABITACIONHOSTAL	NORMAL	ISIS2304C19202310	A_HABITACIONHOSTAL	TA
16	ISIS2304C19202310	A_HABITACIONHUESPED_PK	NORMAL	ISIS2304C19202310	A_HABITACIONHUESPED	TA
17	ISIS2304C19202310	IDX_RESERVA_FECHAINI_IDALOJAMIENTO	FUNCTION-BASED NORMAL	ISIS2304C19202310	A_RESERVA	TA
18	ISIS2304C19202310	A_APARTAMENTOALQUILER_PK	NORMAL	ISIS2304C19202310	A_APARTAMENTOALQUILER	TA

En la imagen anterior podemos observar todos los índices usados en nuestra base de datos, los índices creados por nosotros fueron:

- Idx_reserva_fechaini: En la tabla Reserva y dado por el atributo Fecha_Ini
- idx_alojamiento_id_idoperador: En la tabla Reserva y dado por (Fecha_ini, idAlojamiento)
- idx_alojamiento_id_idoperador: En la tabla Alojamiento y dado por (ID, IDOPERADOR)
- Idx_reserva_ganancia: En la tabla Reserva y dado por (GANANCIA)

Principalmente, estos índices fueron creados para el requerimiento de consulta número 12, dado que este fue el único requerimiento en el cual los índices creados por Oracle no servían para garantizar la eficiencia en las consultas. Se entrar más a detalle sobre la creación y el tipo de estos índices en la sección del RFC12. No fue necesario crear más índices dado que el resto de las consultas cumplían con el desempeño esperado. De manera que, consideramos que la creación de otros índices simplemente resultaría dando un efecto negativo a la base de datos en términos de eficiencia y espacio ocupado.

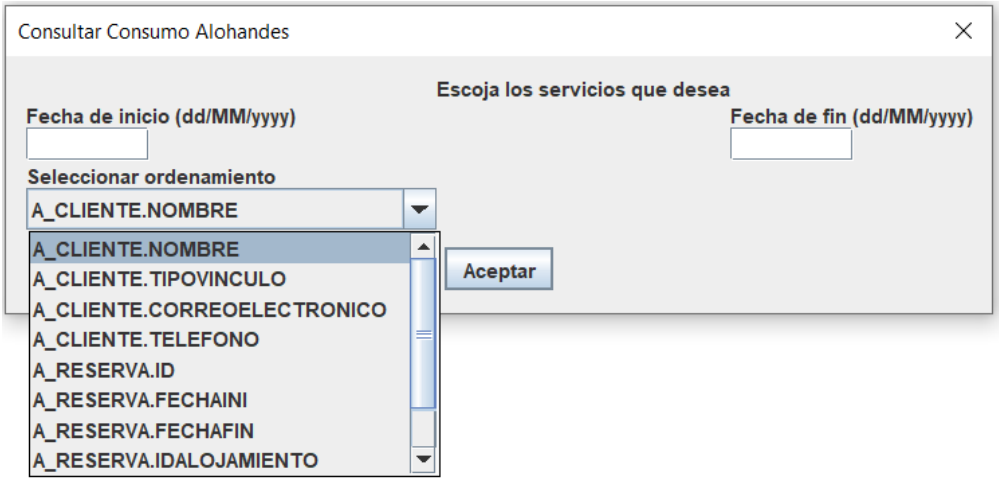
En general, la mayoría de los índices fueron creados de forma automática por Oracle, basado en las PK de cada tabla. Se hablará sobre la utilidad de cada uno de estos a continuación, enfocados en cada consulta.

3.3. RFC10 - CONSULTAR CONSUMO EN ALOHANDES

3.3.1. Sentencia SQL

```
SELECT A_CLIENTE.IDENTIFICACION IDENTIFICACION,
A_CLIENTE.NOMBRE NOMBRE,
A_CLIENTE.TIPOVINCULO TIPOVINCULO,
A_CLIENTE.CORREOELECTRONICO CORREOELECTRONICO,
A_CLIENTE.TELEFONO TELEFONO,
A_RESERVA.ID ID,
A_RESERVA.FECHAINI FECHAINI,
A_RESERVA.FECHAFIN FECHAFIN,
A_RESERVA.IDALOJAMIENTO IDALOJAMIENTO,
A_RESERVA.ESTADO ESTADO
FROM A_CLIENTE, A_RESERVA, A_ALOJAMIENTO
WHERE A_RESERVA.IDALOJAMIENTO = A_ALOJAMIENTO.ID
    AND A_CLIENTE.IDENTIFICACION = A_RESERVA.IDENTIFICACIONCLIENTE
    AND A_RESERVA.FECHAINI BETWEEN '01/01/2022' AND '01/01/2023'
    AND A_RESERVA.FECHAFIN BETWEEN '01/01/2022' AND '01/01/2023'
GROUP BY A_CLIENTE.IDENTIFICACION ,
A_CLIENTE.NOMBRE ,
A_CLIENTE.TIPOVINCULO ,
A_CLIENTE.CORREOELECTRONICO ,
A_CLIENTE.TELEFONO ,
A_RESERVA.ID ,
A_RESERVA.FECHAINI ,
A_RESERVA.FECHAFIN ,
A_RESERVA.IDALOJAMIENTO ,
A_RESERVA.ESTADO ,
A_ALOJAMIENTO.TIPOALOJAMIENTO
ORDER BY A_CLIENTE.IDENTIFICACION
FETCH FIRST 100 ROWS ONLY;
```

En este requerimiento se solicitaba conocer la información de los usuarios que realizaron al menos una reserva de una determinada oferta de alojamiento en un rango de fechas. Se le da la opción al usuario de clasificar los resultados mediante agrupamiento y ordenamiento según sean sus intereses.



3.3.2. Plan de consulta y análisis

Plan de ejecución del DBMS:

```
Plan hash value: 766149783

-------------------------------------------------------------------------------------------------
| Id  | Operation                | Name           | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                |   100 | 96000 |       |  8607   (1)| 00:00:01 |
|*  1 |  VIEW                    |                |   100 | 96000 |       |  8607   (1)| 00:00:01 |
|*  2 |   WINDOW SORT PUSHED RANK|                |   102K|    13M|    15M|  8607   (1)| 00:00:01 |
|   3 |    HASH GROUP BY         |                |   102K|    13M|    15M|  8607   (1)| 00:00:01 |
|*  4 |     HASH JOIN            |                |   102K|    13M|  7920K|  2355   (1)| 00:00:01 |
|*  5 |      HASH JOIN           |                |   102K|  6713K|  5416K|  1368   (2)| 00:00:01 |
|*  6 |       TABLE ACCESS FULL  | A_RESERVA      |   102K|  4208K|       |   226   (6)| 00:00:01 |
|   7 |       TABLE ACCESS FULL  | A_ALOJAMIENTO  |   270K|  6591K|       |   405   (1)| 00:00:01 |
|   8 |      TABLE ACCESS FULL   | A_CLIENTE      |   100K|  6835K|       |   214   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("from$_subquery$_004"."rowlimit_$$_rownumber"<=100)
   2 - filter(ROW_NUMBER() OVER ( ORDER BY "A_CLIENTE"."IDENTIFICACION")<=100)
   4 - access("A_CLIENTE"."IDENTIFICACION"="A_RESERVA"."IDENTIFICACIONCLIENTE")
   5 - access("A_RESERVA"."IDALOJAMIENTO"="A_ALOJAMIENTO"."ID")
```

Propuesta Plan de Ejecución:

Vamos a comenzar con la tabla A_RESERVA, aplicando un filtro sigma para seleccionar solo las entradas que estén dentro de las fechas especificadas por el usuario.

Posteriormente, ejecutaremos una operación de "inner join" entre las tablas A_CLIENTE, A_RESERVA y A_ALOJAMIENTO. Teniendo en cuenta las limitaciones de hardware que podríamos enfrentar, suponemos que los datos de estas tablas (con alrededor de 520,000 registros) se ajustarán a la memoria principal, ya que es muy poco probable que excedan 1 GB de RAM. En este contexto, optaremos por el método de ONE-PASS Join.

Una vez realizado el join, procederemos a agrupar los datos de la siguiente manera:

- A_CLIENTE.NOMBRE
- A_CLIENTE.TIPOVINCULO
- A_CLIENTE.CORREOELECTRONICO
- A_CLIENTE.TELEFONO
- A_RESERVA.ID
- A_RESERVA.FECHAINI
- A_RESERVA.FECHAFIN
- A_RESERVA.IDALOJAMIENTO
- A_RESERVA.ESTADO
- A_ALOJAMIENTO.TIPOALOJAMIENTO
- A_CLIENTE.IDENTIFICACION

Posteriormente, clasificaremos los resultados de acuerdo con las preferencias del usuario y seleccionaremos los primeros 100 registros.

Finalmente, proyectaremos los resultados finales, que incluirán la siguiente información:

- A_CLIENTE.NOMBRE
- A_CLIENTE.TIPOVINCULO
- A_CLIENTE.CORREOELECTRONICO
- A_CLIENTE.TELEFONO
- A_RESERVA.ID
- A_RESERVA.FECHAINI
- A_RESERVA.FECHAFIN
- A_RESERVA.IDALOJAMIENTO
- A_RESERVA.ESTADO
- A_ALOJAMIENTO.TIPOALOJAMIENTO
- A_CLIENTE.IDENTIFICACION

3.3.3. Uso de índices

Para este requerimiento podemos observar que se realiza un producto cruz entre Reserva, Alojamiento y Cliente, de manera que, resulta útil el uso de índices creados automáticamente como lo son: A_ALOJAMIENTO_PK, A_RESERVA_PK y A_CLIENTE_PK.

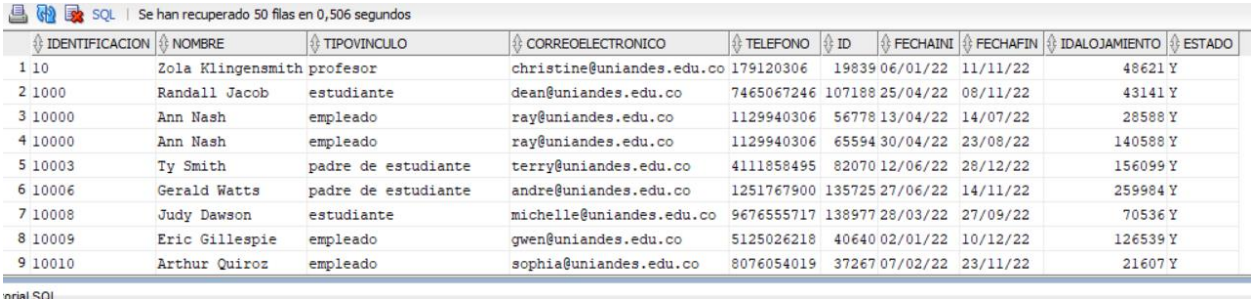
En vista de que con estos índices se cumple el requerimiento de eficiencia solicitado, optamos por no crear más, dado que, de hacerlo podría resultar perjudicial, puesto que podría haber un desperdicio de espacio y se podría empeorar el tiempo de consulta.

3.3.4. Tiempo de ejecución y distribución de los datos

La carga de datos para este requerimiento sigue la mencionada en el punto 3.1. La distribución de datos que se tiene es:

- Para Alojamiento: 27%, es decir 270.000 datos.
- Para Reserva: 15%, es decir 150.000 datos
- Para cliente: 10%, o sea, 100.000 datos

Lo cual nos da que en esta consulta se manejaron entre todas las operaciones un 52% del total de los datos, es decir 520.000 datos, para los cuales obtuvimos los siguientes resultados:


SQL | Se han recuperado 50 filas en 0,506 segundos

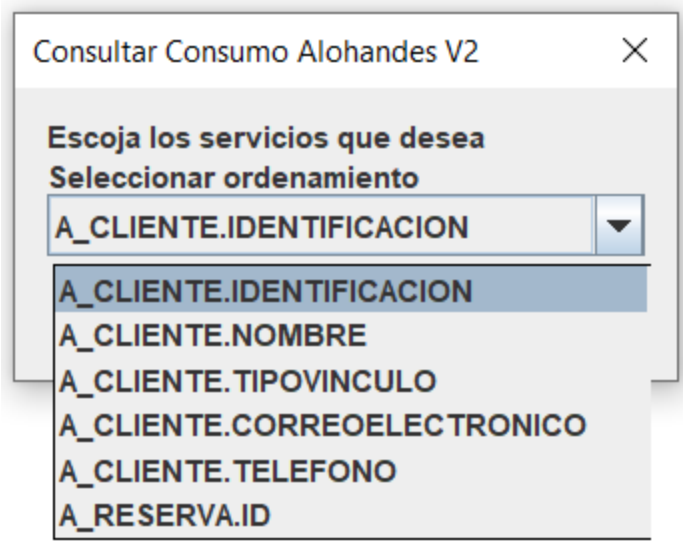
	IDENTIFICACION	NOMBRE	TIPOVINCULO	CORREOELECTRONICO	TELEFONO	ID	FECHAINI	FECHAFIN	IDALOJAMIENTO	ESTADO
1	10	Zola Klingensmith	profesor	christine@uniandes.edu.co	179120306	19839	06/01/22	11/11/22	48621	Y
2	1000	Randall Jacob	estudiante	dean@uniandes.edu.co	7465067246	107188	25/04/22	08/11/22	43141	Y
3	10000	Ann Nash	empleado	ray@uniandes.edu.co	1129940306	56778	13/04/22	14/07/22	28588	Y
4	10000	Ann Nash	empleado	ray@uniandes.edu.co	1129940306	65594	30/04/22	23/08/22	140588	Y
5	10003	Ty Smith	padre de estudiante	terry@uniandes.edu.co	4111858495	82070	12/06/22	28/12/22	156099	Y
6	10006	Gerald Watts	padre de estudiante	andre@uniandes.edu.co	1251767900	135725	27/06/22	14/11/22	259984	Y
7	10008	Judy Dawson	estudiante	michelle@uniandes.edu.co	9676555717	138977	28/03/22	27/09/22	70536	Y
8	10009	Eric Gillespie	empleado	gwen@uniandes.edu.co	5125026218	40640	02/01/22	10/12/22	126539	Y
9	10010	Arthur Quiroz	empleado	sophia@uniandes.edu.co	8076054019	37267	07/02/22	23/11/22	21607	Y

3.4. RFC11 - CONSULTAR CONSUMO EN ALOHANDES – RFC10-V2

3.4.1. Sentencia SQL

```
SELECT A_CLIENTE.IDENTIFICACION IDENTIFICACION,
A_CLIENTE.NOMBRE NOMBRE,
A_CLIENTE.TIPOVINCULO TIPOVINCULO,
A_CLIENTE.CORREOELECTRONICO CORREOELECTRONICO,
A_CLIENTE.TELEFONO TELEFONO,
A_RESERVA.ID ID
FROM A_CLIENTE LEFT JOIN A_RESERVA ON A_CLIENTE.IDENTIFICACION = A_RESERVA.IDENTIFICACIONCLIENTE
WHERE A_RESERVA.IDENTIFICACIONCLIENTE IS NULL
GROUP BY A_CLIENTE.IDENTIFICACION ,
A_CLIENTE.NOMBRE ,
A_CLIENTE.TIPOVINCULO ,
A_CLIENTE.CORREOELECTRONICO ,
A_CLIENTE.TELEFONO ,
A_RESERVA.ID
ORDER BY A_CLIENTE.IDENTIFICACION;
```

Este requerimiento busca consultar los usuarios que no han reservado una oferta de alojamiento en un rango de fechas. La consulta debe permitir agrupar y ordenar los resultados según los intereses del usuario, como datos del cliente, oferta de alojamiento y tipo de alojamiento.



3.4.2. Planes de consulta y análisis

Plan de ejecución del DBMS:

PLAN_TABLE_OUTPUT

```
Plan hash value: 1971816188

-------------------------------------------------------------------------------------------------
| Id  | Operation                | Name       | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |            |   100K|  7910K|       |  4773   (1)| 00:00:01 |
|   1 |  SORT ORDER BY           |            |   100K|  7910K|  9104K|  4773   (1)| 00:00:01 |
|   2 |   HASH GROUP BY          |            |   100K|  7910K|  9104K|  4773   (1)| 00:00:01 |
|*  3 |    FILTER                |            |       |       |       |            |          |
|*  4 |     HASH JOIN RIGHT OUTER|            |   100K|  7910K|  3376K|   982   (1)| 00:00:01 |
|   5 |      TABLE ACCESS FULL   | A_RESERVA  |   150K|  1611K|       |   214   (1)| 00:00:01 |
|   6 |      TABLE ACCESS FULL   | A_CLIENTE  |   100K|  6835K|       |   214   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("A_RESERVA"."IDENTIFICACIONCLIENTE" IS NULL)
   4 - access("A_CLIENTE"."IDENTIFICACION"="A_RESERVA"."IDENTIFICACIONCLIENTE"(+))
```

Propuesta Plan de Ejecución:

Para implementar este requerimiento funcional, comenzaremos con la ejecución del plan establecido. Inicialmente, realizaremos un Join entre las tablas A_CLIENTE y A_RESERVA.

Este proceso es ligero y suponemos que cabrá en memoria, dado que manejamos aproximadamente 250.000 registros, lo cual es menos de 1GB en memoria RAM. Con base en esta suposición, podremos utilizar un ONE-PASS Join para fusionar las tablas mencionadas anteriormente.

Posteriormente, aplicaremos un filtro utilizando una operación sigma para buscar los registros en los que el atributo 'identificación clientes' de la tabla A_RESERVA sea nulo. No obstante, debemos tener en cuenta que SQL no soporta la operación de igualdad con null, por lo que usaremos la operación IS NULL en su lugar.

Tras este filtrado, procederemos a agrupar los datos utilizando los siguientes campos:

- A_CLIENTE.IDENTIFICACION
- A_CLIENTE.NOMBRE
- A_CLIENTE.TIPOVINCULO
- A_CLIENTE.CORREOELECTRONICO
- A_CLIENTE.TELEFONO
- A_RESERVA.ID

A continuación, se realizará el ordenamiento de los datos conforme a las preferencias del usuario.

Finalmente, efectuaremos la proyección final de los datos que resultará en la presentación de los siguientes campos:

- A_CLIENTE.IDENTIFICACION
- A_CLIENTE.NOMBRE
- A_CLIENTE.TIPOVINCULO
- A_CLIENTE.CORREOELECTRONICO
- A_CLIENTE.TELEFONO
- A_RESERVA.ID

3.4.3. Uso de índices

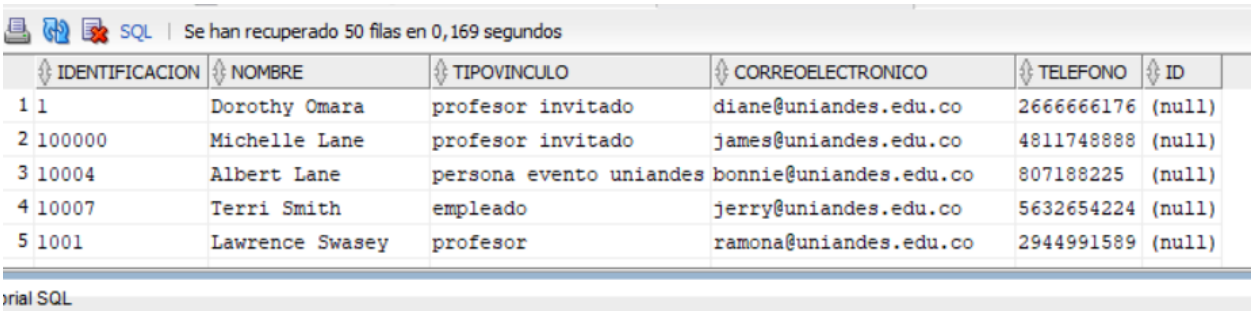
En este requerimiento observamos que se hace un LEFT JOIN entre cliente y reserva, de manera que los índices usados son los creados automáticamente por Oracle, específicamente: A_CLIENTE_PK y A_RESERVA_PK.

En un principio intentamos crear nuevos índices para experimentar, lo cual nos llevo a concluir varias cosas. En primer lugar, los índices creados automáticamente sobre las PK de cada tabla fueron bastante útiles en la eficiencia de la consulta. Por otra parte, los índices que creamos daban resultados de tiempo peores, y consumían espacio dentro de la base de datos.

3.4.4. Tiempo de ejecución y distribución de datos

En lo que respecta a la distribución de los datos, evidenciamos que en la consulta se usan Cliente y Reserva. Así siguiendo el modelo de distribución de datos que tenemos, Cliente tiene un 10%, es decir 100.000 datos y Reserva posee un 15%, o sea 150.000 datos.

Esto nos da que la consulta maneja alrededor de un 25% de todos los datos, o sea, 250.000 datos para un resultado de:



SQL | Se han recuperado 50 filas en 0,169 segundos

	IDENTIFICACION	NOMBRE	TIPOVINCULO	CORREOELECTRONICO	TELEFONO	ID
1	1	Dorothy Omara	profesor invitado	diane@uniandes.edu.co	2666666176	(null)
2	100000	Michelle Lane	profesor invitado	james@uniandes.edu.co	4811748888	(null)
3	10004	Albert Lane	persona evento uniandes	bonnie@uniandes.edu.co	807188225	(null)
4	10007	Terri Smith	empleado	jerry@uniandes.edu.co	5632654224	(null)
5	1001	Lawrence Swasey	profesor	ramona@uniandes.edu.co	2944991589	(null)

orial SQL

3.5. RFC12 - CONSULTAR FUNCIONAMIENTO

3.5.1. Sentencia SQL

```
SELECT SEMANA,
       CANTIDAD_RESERVAS,
       MAX(NUMOCUPAMIENTO) KEEP (DENSE_RANK FIRST ORDER BY RANK_MAX_OCUPACION) AS MAX_OCUPACION,
       MAX(IDALOJAMIENTO) KEEP (DENSE_RANK FIRST ORDER BY RANK_MAX_OCUPACION) AS MAX_ID_ALOJAMIENTO,
       MIN(NUMOCUPAMIENTO) KEEP (DENSE_RANK FIRST ORDER BY RANK_MIN_OCUPACION) AS MIN_OCUPACION,
       MIN(IDALOJAMIENTO) KEEP (DENSE_RANK FIRST ORDER BY RANK_MIN_OCUPACION) AS MIN_ID_ALOJAMIENTO,
       MIN(OPERADOR_COUNT) AS NUMSOLMIN,
       MAX(IDOPERADOR) KEEP (DENSE_RANK FIRST ORDER BY OPERADOR_COUNT) AS OP_MIN_SOLICITADO,
       MAX(OPERADOR_COUNT) AS NUMSOLMAX,
       MAX(IDOPERADOR) KEEP (DENSE_RANK LAST ORDER BY OPERADOR_COUNT) AS OP_MAX_SOLICITADO
FROM (
  SELECT S.SEMANA,
         COUNT(R.FECHAINI) OVER (PARTITION BY S.SEMANA) AS CANTIDAD_RESERVAS,
         R.NUMOCUPAMIENTO,
         R.IDALOJAMIENTO,
         A.IDOPERADOR,
         RANK() OVER (PARTITION BY S.SEMANA ORDER BY R.NUMOCUPAMIENTO DESC) AS RANK_MAX_OCUPACION,
         RANK() OVER (PARTITION BY S.SEMANA ORDER BY R.NUMOCUPAMIENTO) AS RANK_MIN_OCUPACION,
         COUNT(A.IDOPERADOR) OVER (PARTITION BY S.SEMANA, A.IDOPERADOR) AS OPERADOR_COUNT
  FROM (
    SELECT TO_DATE('01/01/2022', 'DD/MM/YYYY') + ((ROWNUM - 1) * 7) AS SEMANA
    FROM DUAL
    CONNECT BY ROWNUM <= 52
  ) S
  LEFT JOIN A_RESERVA R ON TRUNC(R.FECHAINI, 'IW') = TRUNC(S.SEMANA, 'IW')
  LEFT JOIN A_ALOJAMIENTO A ON R.IDALOJAMIENTO = A.ID
)
```

```
GROUP BY SEMANA, CANTIDAD_RESERVAS
ORDER BY SEMANA;
```

En este requerimiento se solicitaba ver para cada semana del año la oferta de alojamiento con más y menos ocupación, y el operador más y menos solicitado.

3.5.2. Planes de consulta y análisis

Plan de ejecución del DBMS:

```
PLAN_TABLE_OUTPUT
Plan hash value: 3108682971

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                |  1500 |   312K|   628   (2)| 00:00:01 |
|   1 |  SORT ORDER BY                     |                |  1500 |   312K|   628   (2)| 00:00:01 |
|   2 |   SORT GROUP BY                    |                |  1500 |   312K|   628   (2)| 00:00:01 |
|   3 |    VIEW                            |                |  1500 |   312K|   626   (2)| 00:00:01 |
|   4 |     WINDOW SORT                    |                |  1500 | 49500 |   626   (2)| 00:00:01 |
|   5 |      WINDOW SORT                   |                |  1500 | 49500 |   626   (2)| 00:00:01 |
|   6 |       WINDOW SORT                  |                |  1500 | 49500 |   626   (2)| 00:00:01 |
|*  7 |        HASH JOIN OUTER             |                |  1500 | 49500 |   623   (1)| 00:00:01 |
|*  8 |         HASH JOIN OUTER            |                |  1500 | 33000 |   217   (1)| 00:00:01 |
|   9 |          VIEW                      |                |     1 |     6 |     2   (0)| 00:00:01 |
|  10 |           COUNT                    |                |       |       |            |          |
|* 11 |            CONNECT BY WITHOUT FILTERING|            |       |       |            |          |
|  12 |             FAST DUAL              |                |     1 |       |     2   (0)| 00:00:01 |
|  13 |          TABLE ACCESS FULL         | A_RESERVA      |  150K|  2343K|   215   (1)| 00:00:01 |
|  14 |         TABLE ACCESS FULL          | A_ALOJAMIENTO  |  270K|  2900K|   405   (1)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------
```

Propuesta plan de ejecución:

Para la implementación de este requerimiento proponemos realizar lo siguiente:

- Generar la lista de las 52 semanas usando la función connect by.
- Realizar el LEFT JOIN entre la Reserva y Alojamiento.
- Calculamos la cantidad de reservas por medio de la función COUNT, así sabemos la cantidad de reservas por semana.
- Calculamos la cantidad de ocupación para el alojamiento con más ocupación esto por medio de la función MAX, en unión con KEEP DENSE_RANK FIRST, lo cual nos permite hacer un ranking y obtener el alojamiento que tiene más ocupación de esa semana junto con su id.
- Realizamos el mismo paso anterior, pero para el alojamiento con menor ocupación.

- Ahora, calculamos el número de reservas que tiene cada operador por semana, esto nos permite saber cual fue el operador más solicitado. Usamos MIN para saber el cual es el operador con menos reservas de esa semana.
- Se consulta el operador más solicitado por cada semana, esto lo hacemos con la función MAX, así, se obtiene cual fue el operador que tuvo más reservas, y su respectivo id.
- Finalmente, se realiza el agrupamiento y ordenamiento de los resultados, estos los agrupamos y ordenamos por semana.

3.5.3. Uso de índices

Para esta consulta en un principio consideramos usar solo los índices creados automáticamente por Oracle(A_CLIENTE_PK, A_RESERVA_PK y A_ALOJAMIENTO_PK). Sin embargo, luego de pruebas y análisis determinamos que eso no era suficiente para garantizar el requerimiento de eficiencia solicitado, de manera que optamos por crear los siguientes índices:

```
CREATE INDEX idx_reserva_fechaini ON A_RESERVA(TRUNC(FECHAINI, 'IW'));
CREATE INDEX idx_alojamiento_id_idoperador ON A_ALOJAMIENTO(ID, IDOPERADOR);
CREATE INDEX idx_reserva_fechaini_idalojamiento ON A_RESERVA(TRUNC(FECHAINI, 'IW'), IDALOJAMIENTO);
```

Fueron de gran ayuda, principalmente de Reserva, dado que la consulta se realizaba por cada semana del año, y la fecha inicial de la reserva se convierte al formato de semana.

3.5.4. Tiempo de ejecución y distribución de datos

Puesto que en la consulta se usa Cliente, Reserva y Alojamiento se tiene una distribución del 52% del total de los datos, es decir 520.000 datos.

Tiempo con índices:

SQL | Se han recuperado 50 filas en 0,958 segundos

	SEMANA	CANTIDAD_RESERVAS	MAX_OCUPACION	MAX_ID_ALOJAMIENTO	MIN_OCUPACION	MIN_ID_ALOJAMIENTO	NUMSOLMIN	OP_MIN_SOLICITADO	NUMSOLMAX	OP_MAX_SOLICITADO
1	01/01/22	1678	10	269622	1	1747	1	9974	2	9647
2	08/01/22	5958	10	269986	1	417	1	9998	4	16401
3	15/01/22	5833	10	269576	1	1833	1	9996	4	4047
4	22/01/22	5920	10	269964	1	112	1	9999	4	6717
5	29/01/22	5788	10	269926	1	239	1	9990	4	17636
6	05/02/22	5730	10	269722	1	183	1	9998	4	16470
7	12/02/22	5817	10	269837	1	73	1	9996	4	2496
8	19/02/22	5756	10	269536	1	116	1	9999	4	12562

Tiempo sin índices:

SQL | Se han recuperado 50 filas en 1,483 segundos

	SEMANA	CANTIDAD_RESERVAS	MAX_OCUPACION	MAX_ID_ALOJAMIENTO	MIN_OCUPACION	MIN_ID_ALOJAMIENTO	NUMSOLMIN	OP_MIN_SOLICITADO	NUMSOLMAX	OP_MAX_SOLICITADO
1	01/01/22	1678	10	269622	1	1747	1	9974	2	9647
2	08/01/22	5958	10	269986	1	417	1	9998	4	16401
3	15/01/22	5833	10	269576	1	1833	1	9996	4	4047
4	22/01/22	5920	10	269964	1	112	1	9999	4	6717
5	29/01/22	5788	10	269926	1	239	1	9990	4	17636

rial SQL

3.6. RFC13 - CONSULTAR LOS BUENOS CLIENTES

3.6.1. Sentencia SQL

```
SELECT
    a_cliente.IDENTIFICACION,
    a_cliente.NOMBRE,
    a_cliente.TIPOVINCULO,
    a_cliente.CORREOELECTRONICO,
    a_cliente.TELEFONO,
    LISTAGG(criteria.CRITERIO, ', ') WITHIN GROUP (ORDER BY criteria.CRITERIO) AS CRITERIOS
FROM
    a_cliente
    JOIN (
        SELECT DISTINCT
            a_cliente.IDENTIFICACION,
            CASE
                WHEN COUNT(DISTINCT TRUNC(a_reserva.FECHAINI, 'MM')) >= 3 THEN 'Reserva mensual'
                WHEN AVG(a_reserva.GANANCIA) > 150 THEN 'Alojamientos costosos'
                WHEN COUNT(DISTINCT CASE WHEN a_alojamiento.TIPOALOJAMIENTO = 'HabitacionHotel' AND a_habitacionhotel.TIPOHABITACION = 'suite' THEN TRUNC(a_reserva.FECHAINI, 'MM') END) >= 1 THEN 'Reserva en suite'
                ELSE NULL
            END AS CRITERIO
        FROM
            a_cliente
            JOIN a_reserva ON a_cliente.IDENTIFICACION = a_reserva.IDENTIFICACIONCLIENTE
            JOIN a_alojamiento ON a_reserva.IDALOJAMIENTO = a_alojamiento.ID
            LEFT JOIN a_habitacionhotel ON a_alojamiento.ID = a_habitacionhotel.ID
        WHERE
            a_reserva.FECHAINI BETWEEN ADD_MONTHS(SYSDATE, -3) AND SYSDATE
            OR a_reserva.FECHAFIN BETWEEN ADD_MONTHS(SYSDATE, -3) AND SYSDATE
        GROUP BY
            a_cliente.IDENTIFICACION
    ) criteria ON a_cliente.IDENTIFICACION = criteria.IDENTIFICACION
WHERE
    criteria.CRITERIO IS NOT NULL
GROUP BY
    a_cliente.IDENTIFICACION,
    a_cliente.NOMBRE,
    a_cliente.TIPOVINCULO,
    a_cliente.CORREOELECTRONICO,
    a_cliente.TELEFONO
FETCH FIRST 100 ROWS ONLY    ;
```

Este requerimiento busca consultar a los buenos clientes de AlohAndes. Estos clientes se dividen en tres categorías: aquellos que hacen al menos una reserva al mes, aquellos que siempre reservan alojamientos costosos (más de USD 150 por noche) y aquellos que siempre reservan suites. La consulta debe devolver toda la información de estos clientes, incluyendo los motivos que justifican su clasificación como "buenos clientes". Solo el Administrador de AlohAndes puede realizar esta operación.

3.6.2. Planes de consulta y análisis

Plan de ejecución del DBMS:

```
Plan hash value: 940619105

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                   |   100 |  259K|   677   (3)| 00:00:01 |
|*  1 |  VIEW                         |                   |   100 |  259K|   677   (3)| 00:00:01 |
|*  2 |   WINDOW NOSORT STOPKEY       |                   |     1 |   88 |   677   (3)| 00:00:01 |
|   3 |    SORT GROUP BY              |                   |     1 |   88 |   677   (3)| 00:00:01 |
|   4 |     NESTED LOOPS              |                   |     1 |   88 |   676   (2)| 00:00:01 |
|   5 |      NESTED LOOPS             |                   |     1 |   88 |   676   (2)| 00:00:01 |
|   6 |       VIEW                    |                   |     1 |   18 |   675   (2)| 00:00:01 |
|*  7 |        FILTER                 |                   |       |      |            |          |
|   8 |         SORT GROUP BY         |                   |     1 |   79 |   675   (2)| 00:00:01 |
|*  9 |          HASH JOIN RIGHT OUTER|                   | 16023 | 1236K|   673   (2)| 00:00:01 |
|  10 |           TABLE ACCESS FULL   | A_HABITACIONHOTEL | 40000 |  546K|    47   (0)| 00:00:01 |
|* 11 |           HASH JOIN           |                   | 16023 | 1017K|   626   (2)| 00:00:01 |
|* 12 |            TABLE ACCESS FULL  | A_RESERVA         | 16023 |  625K|   220   (4)| 00:00:01 |
|  13 |            TABLE ACCESS FULL  | A_ALOJAMIENTO     |  270K| 6591K|   405   (1)| 00:00:01 |
|* 14 |       INDEX UNIQUE SCAN       | A_CLIENTE_PK      |     1 |      |     0   (0)| 00:00:01 |
|  15 |      TABLE ACCESS BY INDEX ROWID| A_CLIENTE       |     1 |   70 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------
```

Propuesta Plan de Ejecución:

Inicialmente, se realiza una operación de join entre la tabla A_CLIENTE y una subconsulta, la cual contiene los criterios que cada cliente ha cumplido. Dado que las tablas implicadas pueden caber en la memoria, es viable llevar a cabo un ONE-PASS join para mejorar la eficiencia del proceso.

A continuación, se filtran los resultados de la unión previa. Para ello, se aplica un operador sigma con el fin de eliminar las filas donde el criterio sea null, lo que garantiza la exclusión de clientes sin criterios definidos en los resultados finales.

Posteriormente, se realiza una operación de agrupación de los datos resultantes basada en los siguientes campos: a_cliente.IDENTIFICACION, a_cliente.NOMBRE, a_cliente.TIPOVINCULO, a_cliente.CORREOELECTRONICO, y a_cliente.TELEFONO.

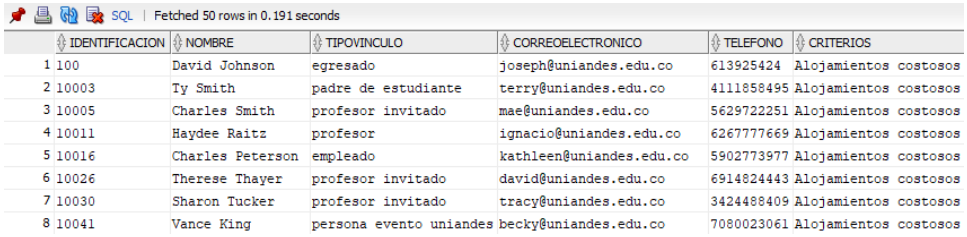
Finalmente, se limitan los resultados a las primeras 100 filas y se proyectan los datos requeridos.

3.6.3. Uso de índices

Para satisfacer este requerimiento funcional, optamos por utilizar los índices proporcionados por el sistema de gestión de base de datos (DBMS), específicamente los índices primarios de las tablas que se basan en índices de hash. Algunos ejemplos que observamos en el plan de ejecución de Oracle son el atributo 'identificacion' para la tabla A_CLIENTE, el atributo 'id' para A_RESERVA, y finalmente, 'id' para A_ALOJAMIENTO.

Además, creímos conveniente establecer un índice adicional para facilitar el rendimiento de este requerimiento funcional. Este nuevo índice, diseñado en la tabla A_RESERVA, se basa en el atributo 'ganancia'. La intención detrás de este índice es permitir consultas de rango más eficientes, como las que realizamos en nuestra consulta. Este índice se define como un árbol B+ denso.

3.6.4. Tiempo de ejecución y distribución de datos



SQL | Fetched 50 rows in 0.191 seconds

	IDENTIFICACION	NOMBRE	TIPOVINCULO	CORREOELECTRONICO	TELEFONO	CRITERIOS
1	100	David Johnson	egresado	joseph@uniandes.edu.co	613925424	Alojamientos costosos
2	10003	Ty Smith	padre de estudiante	terry@uniandes.edu.co	4111858495	Alojamientos costosos
3	10005	Charles Smith	profesor invitado	mae@uniandes.edu.co	5629722251	Alojamientos costosos
4	10011	Haydee Raitz	profesor	ignacio@uniandes.edu.co	6267777669	Alojamientos costosos
5	10016	Charles Peterson	empleado	kathleen@uniandes.edu.co	5902773977	Alojamientos costosos
6	10026	Therese Thayer	profesor invitado	david@uniandes.edu.co	6914824443	Alojamientos costosos
7	10030	Sharon Tucker	profesor invitado	tracy@uniandes.edu.co	3424488409	Alojamientos costosos
8	10041	Vance King	persona evento uniandes	becky@uniandes.edu.co	7080023061	Alojamientos costosos

En esta consulta hacemos uso de las tablas A_CLIENTE, A_RESERVA, A_ALOJAMIENTO y A_HABITACIONHOTEL, estas tienen una distribución de datos del 10, 15, 27 y 4 porciento respectivamente lo cual suma a un 56% de los datos usados en el sistema que en este caso será de un millón, lo que daría que este 56% representan 560.000 registros.

4. Construcción de la aplicación, ejecución de pruebas y análisis de resultados

4.1. Documentación carga de datos

Generamos datos mediante un script de Python, construido cuidadosamente para garantizar la facilidad de mantenimiento a largo plazo. Cada tabla de nuestra base de datos se representa con una clase en el script, aprovechando así el paradigma de programación orientada a objetos. Cada una de estas clases tiene su propio método para la generación de datos.

Además, hemos creado una clase "Main", que actúa como un controlador central. Esta clase se encarga de coordinar la generación de datos en todas las demás clases, distribuyendo la cantidad de datos requeridos proporcionalmente según los porcentajes predefinidos.

La ejecución del script es sencilla. Solo se requiere instanciar la clase "Main" y ejecutar la función "generar_datos", proporcionando como parámetro la cantidad de datos que se desea generar. Como resultado, se producirán archivos CSV, que contienen los datos generados y están listos para ser cargados en la base de datos.

Para la carga de datos, hemos utilizado Putty, una terminal. Subimos archivos de control a la web mediante la aplicación Discord, que nos permite alojar archivos de datos. Estos archivos de control especifican la estructura de los datos. Luego, en la consola de Putty, utilizamos el comando "wget" seguido del enlace al archivo de control para descargarlo. Finalmente, cargamos los datos en la base de datos ejecutando el archivo de control con el comando "sqlldr", seguido del nombre de usuario y la contraseña, y la ruta del archivo de control.

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

Realizamos una comparación entre la ejecución de consultas delegadas al manejador de bases de datos Oracle y el uso de instrucciones de control, como "if" y "while", utilizando una consulta SQL simple como podemos ver en la siguiente imagen:

```sql
SELECT A_ALOJAMIENTO.id, A_ALOJAMIENTOSERVICIO.idServicio
FROM A_ALOJAMIENTO
INNER JOIN A_ALOJAMIENTOSERVICIO ON A_ALOJAMIENTO.id = A_ALOJAMIENTOSERVICIO.idAlojamiento
WHERE A_ALOJAMIENTOSERVICIO.idServicio = 2;
```

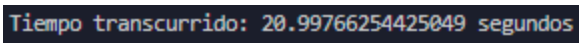
Intentamos replicar la consulta utilizando el lenguaje de programación Python y empleando instrucciones de control, lo cual resultó en el siguiente código:

```python
import csv
import time

def consultar_servicios():
    alojamiento_file = 'Alojamiento.csv'
    servicio_file = 'AlojamientoServicio.csv'
    servicio_id = 2

    registros_servicio = []
    with open(servicio_file, 'r') as file:
        reader = csv.reader(file)
        next(reader)
        for row in reader:
            if int(row[1]) == servicio_id:
                registros_servicio.append(row[0])

    resultados = []
    with open(alojamiento_file, 'r') as file:
        reader = csv.reader(file)
        next(reader)
        for row in reader:
            if row[0] in registros_servicio:
                resultados.append((row[0], row[6]))
    return resultados

start_time = time.time()

resultados = consultar_servicios()

elapsed_time = time.time() - start_time

for resultado in resultados:
    print(f'ID: {resultado[0]}, idServicio: {resultado[1]}')

print(f'Tiempo transcurrido: {elapsed_time} segundos')
```

Tras correr este código de Python nos retornó que se tardaba 20.99 segundos, mientras que el DBMS se tardó 0.014 segundos.

```
Fetched 50 rows in 0.014 seconds
```

Respuesta de la consulta en DBMS Oracle



Respuesta de la consulta realizada en Python

Esta diferencia de tiempos se debe en su mayoría ya que estos sistemas de bases de datos están optimizados para realizar este tipo de consultas y por medio de los planes de ejecución buscaran realizarlo de la manera más optima posible.