

“ITERACION 4”

Raul Rincon y Pedro Lobato
Documento informe de la iteración 4 AlohAndes
Universidad de los Andes, Bogotá, Colombia
{r.rincon, p.lobato}@uniandes.edu.co
Fecha de presentación: Mayo 28 de 2023

Tabla de contenido

1	Introducción	2
2	Modelos	2
2.1	Conceptual	2
2.2	Relacional	2
3	Requerimientos.....	2
	RFC10: Consulta consumo AlohAndes	2
	RFC11: Consulta consumo AlohAndes	6
	RFC12: Consultar funcionamiento.....	10
	RFC13: Consulta buenos clientes.....	11
4	Árbol lógico.....	¡Error! Marcador no definido.
5	Poblado de tablas.....	19
6	Estructura física.....	19
6.1	Selectividad:.....	19
6.2	Índices:	20
6.2.1	23
6.2.2	24

1 Introducción

En el siguiente documento se dará evidencia del proceso y trabajo realizado para la última iteración de sistemas transaccionales, se evidenciarán los cambios, herramientas, diagramas, modelos conceptuales, transaccionales y de negocio utilizados para desarrollar AlohAndes. Se presentará la explicación de los requerimientos de consulta, funcionales y no funcionales más importantes para comprender cómo trabaja la aplicación. Adicionalmente se explicará la carga de datos masivos utilizando una herramienta propia y diferentes técnicas para lograr esta actividad de forma más eficiente y con estructuras nuevas.

2 Modelos

2.1 Conceptual

En esta sección se presenta el modelo de concepto (UML) el cual identifica las entidades relevantes, sus características y las relaciones que se deberían manejar en el negocio

REPOSITORIO

2.2 Relacional

En este apartado se encuentra el modelo de datos relaciones que se genera a partir del modelo conceptual:

REPOSITORIO

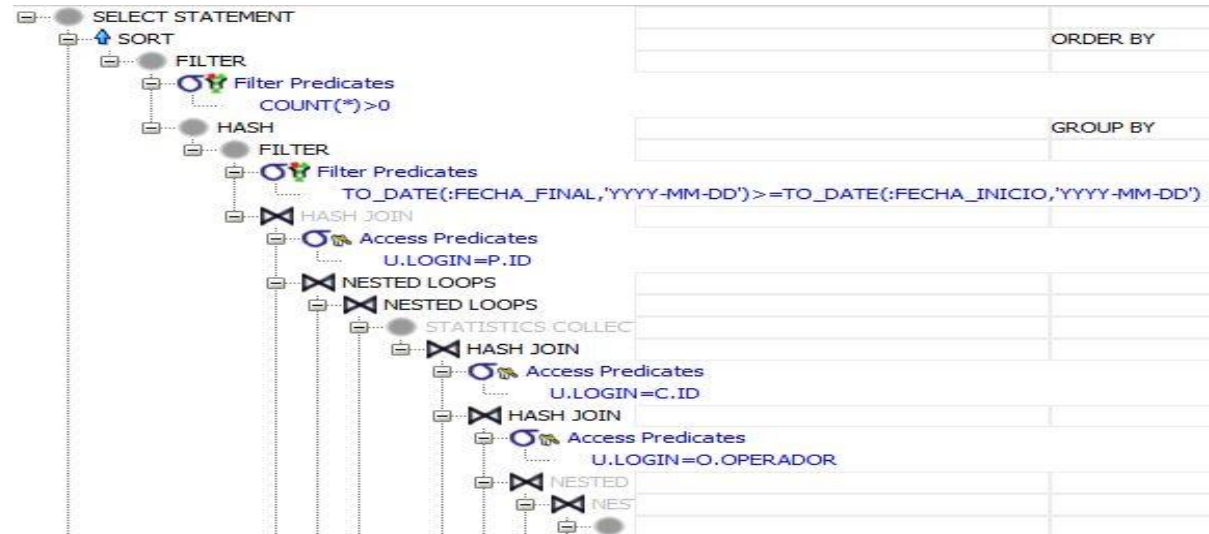
3 Requerimientos

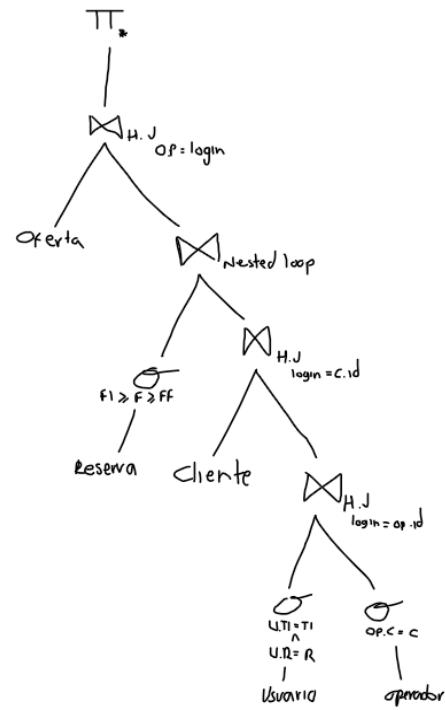
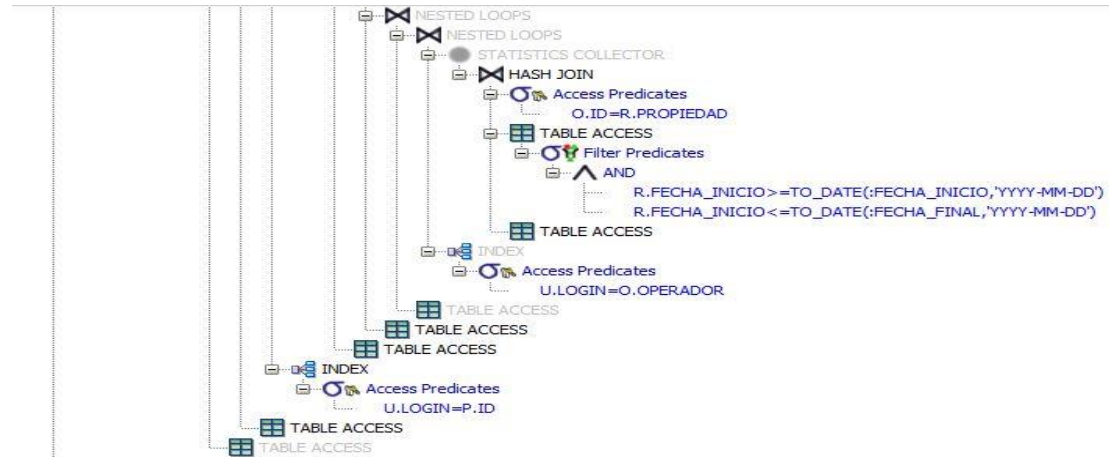
RFC10: Consulta consumo AlohAndes

Para mantener la privacidad se pregunta primero el login del usuario, y luego el id de la propiedad, se busca si la propiedad pertenece al usuario con la siguiente sentencia:

```
SELECT *  
FROM A_Operador o NATURAL INNER JOIN A_PROPIEDAD p  
WHERE o.id = #pIdOp  
ORDER BY #pOrden
```

Consulta Admin:





Los planes realmente cambian ya que Oracle define un orden específico y ciertos Nested Loops mejores los cuales le hacen realizar las operaciones con muchos registros y tablas enormes de forma mucho más sencilla y eficaz. Además de que deja el Nested Loop luego de 3 Hash Join parecido a los planeados, en general un resumen de la query es este y también una explicación del uso de algoritmos en ambos planes:

1. Realizar un join natural (NATURAL JOIN) entre las tablas A_Cliente, A_PROPIEDAD, A_HABITACION, A_APARTAMENTO, A_RESERVAHABITACION y A_RESESRVAAPARTAMENTO utilizando las columnas con los mismos nombres en ambas tablas.
2. Aplicar los siguientes criterios de filtrado utilizando los operadores EQUALS y BETWEEN.
3. Filtrar las filas donde el ID de propiedad (p.id) sea igual al valor proporcionado en el parámetro #pIdProp.
4. Filtrar las filas donde la fecha de inicio de reserva (r.FechaInicio) esté comprendida entre los valores proporcionados en los parámetros #pFI y #pFF.
5. Calcular el número de reservas (reservas) utilizando la función de agregación COUNT() en la columna c.reservas.
6. Filtrar las filas donde el número de reservas (reservas) sea igual a cero.
7. Ordenar los resultados según el criterio especificado en el parámetro #pOrden.

Nested Loops (Bucles anidados):

El método de "nested loops" es eficiente cuando una de las tablas involucradas en el join es pequeña o tiene un índice adecuado.

Oracle realiza un bucle anidado entre las filas de la tabla pequeña y las filas de la tabla grande, buscando coincidencias basadas en las condiciones de join.

Este método es útil cuando la cantidad de datos a combinar es relativamente pequeña y se pueden utilizar índices para acceder rápidamente a los datos relevantes.

El rendimiento de los "nested loops" puede verse afectado si la tabla pequeña es grande o si no hay índices adecuados para acelerar el acceso a los datos.

Hash Joins (Combinaciones hash):

El método de "hash join" se utiliza cuando se deben combinar grandes volúmenes de datos de manera eficiente.

Oracle crea una tabla hash en memoria basada en la tabla de entrada más pequeña y realiza un escaneo de la otra tabla, buscando coincidencias utilizando la tabla hash.

Este método es eficiente para combinar grandes volúmenes de datos porque la tabla hash reduce la cantidad de comparaciones necesarias para encontrar las coincidencias.

Sin embargo, el uso de "hash joins" puede requerir más recursos de memoria en comparación con los "nested loops" y puede tener un rendimiento subóptimo si los datos no caben completamente en la memoria disponible.

TIEMPOS:

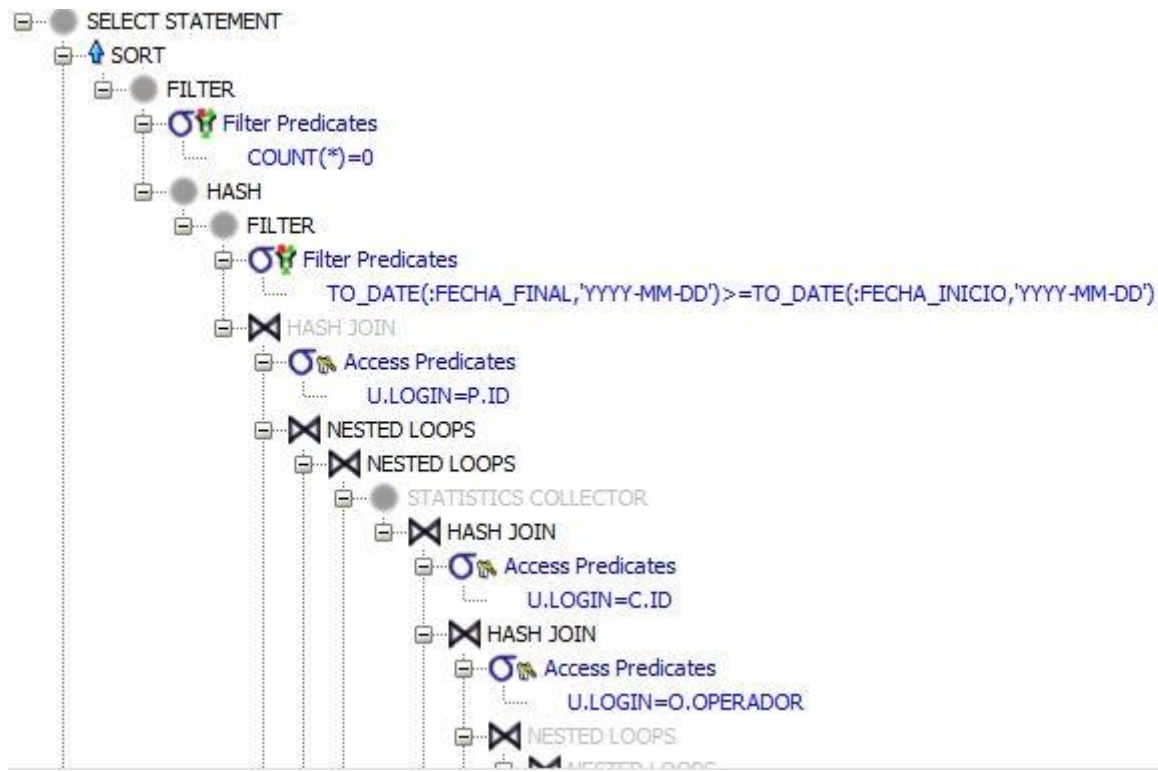
Se han recuperado 50 filas en 0.116 segundos

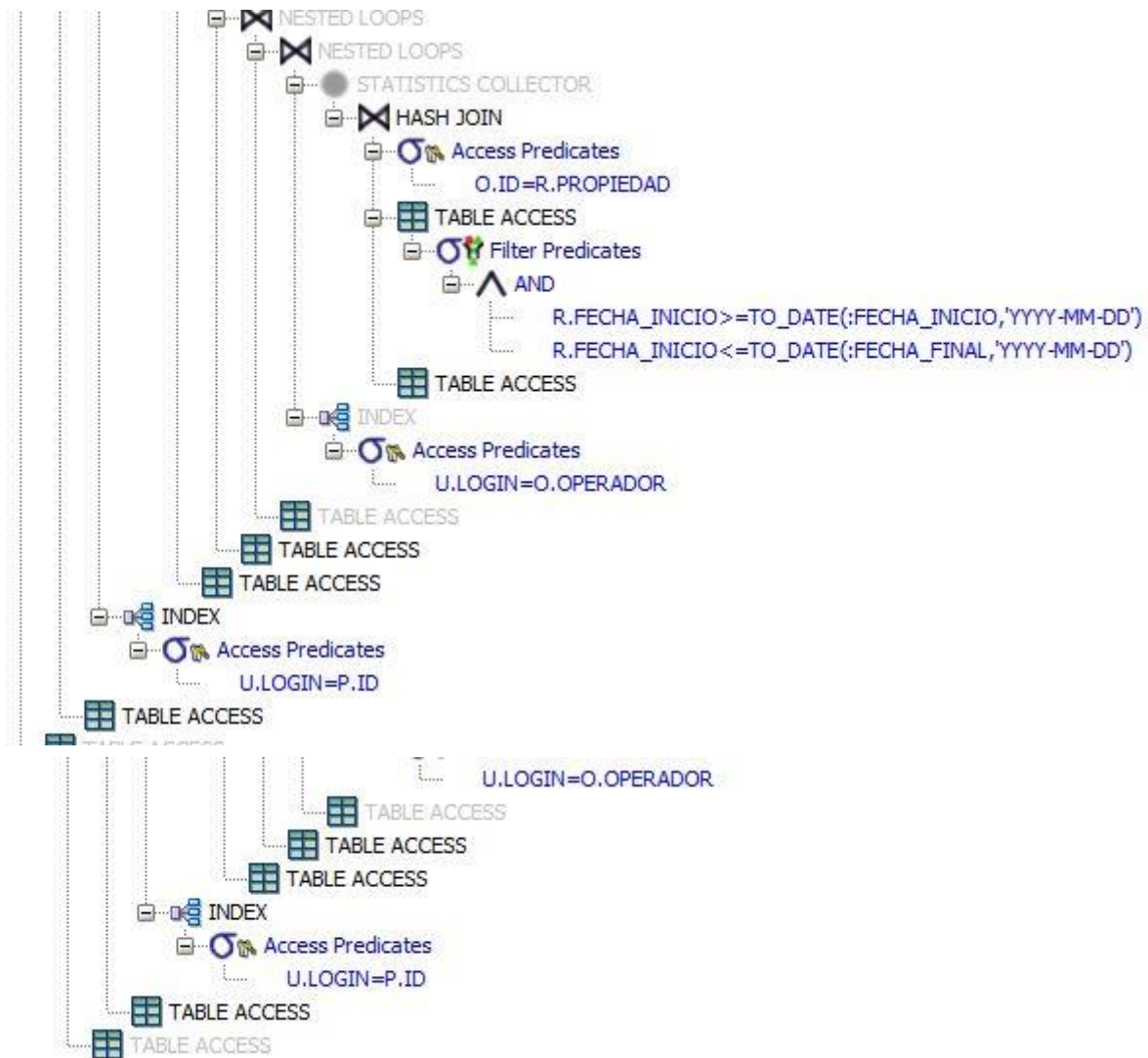
Se han recuperado 50 filas en 0.249 segundos

	USUARIO	TIPO_IDENTIFICACION	RELACION_USUARIO	OFERTA_ID	CAPACIDAD	PRECIO	TIPO_ALOJAMIENTO	CLIENTE_ID	MEDIO_PAGO
1	login100313	CEDULA	ESTUDIANTE	353	75	554	HOSTAL	login100313	CREDITO
2	login49191	CEDULA	ESTUDIANTE	696	40	79	HOSTAL	login49191	EFFECTIVO
3	login137051	CEDULA	ESTUDIANTE	820	31	261	HOSTAL	login137051	DAVIPLATA
4	login137092	CEDULA	ESTUDIANTE	865	41	267	HOSTAL	login137092	EFFECTIVO
5	login17411	CEDULA	ESTUDIANTE	1143	76	952	HOSTAL	login17411	CREDITO
6	login1752	CEDULA	ESTUDIANTE	1263	88	473	HOSTAL	login1752	DEBITO
7	login17721	CEDULA	ESTUDIANTE	1487	54	213	HOSTAL	login17721	CULO

RFC11: Consulta consumo AlohaAndes

```
SELECT count(c.reservas) as res, *
FROM A_Cliente c NATURAL INNER JOIN A_PROPIEDAD p, A_HABITACION h, A_APARTAMENTO a,
A_RESERVAHABITACION rh, A_RESESRVAAPARTAMENTO ra
WHERE p.id = #pIdProp AND r.FechaInicio BETWEEN TO_DATE(#pFI) AND TO_DATE (#pFF) AND res=0
ORDERED BY #pOrden
```





Combinación de tablas:

Se realiza una combinación "natural inner join" entre las tablas A_Cliente, A_PROPIEDAD, A_HABITACION, A_APARTAMENTO, A_RESERVAHABITACION y A_RESESRVAAPARTAMENTO.

Este tipo de join combina las filas de las tablas donde las columnas con el mismo nombre tienen valores coincidentes.

Filtrado de datos:

Se aplica una condición de filtro en la columna p.id para igualarla a #pIdProp, lo cual limita los resultados a una propiedad específica.

Se utiliza la condición r.FechaInicio BETWEEN TO_DATE(#pFI) AND TO_DATE(#pFF) para filtrar las reservas que estén dentro del rango de fechas especificado.

Se agrega una condición res=0, que asumimos que es una columna de la tabla A_Cliente, para filtrar los resultados donde el número de reservas sea igual a cero.

Cálculo y selección de columnas:

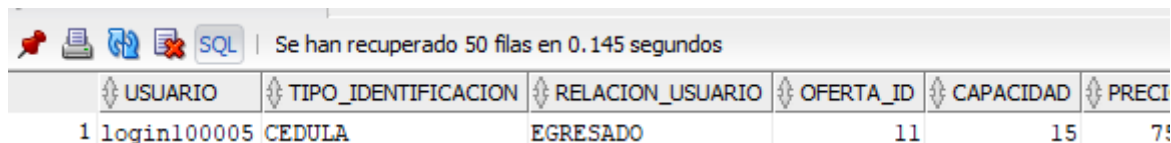
Se utiliza la función count(c.reservas) para calcular el número de reservas y se asigna a la columna con el alias "res".

El operador "*" se utiliza para seleccionar todas las columnas de las tablas involucradas en la consulta.

Ordenamiento:

Se utiliza la cláusula ORDER BY #pOrden para ordenar los resultados según una columna específica.

Realmente esta query funciona y genera el mismo plan de consulta que la query anterior ya que funcionan casi igual o de la misma forma.

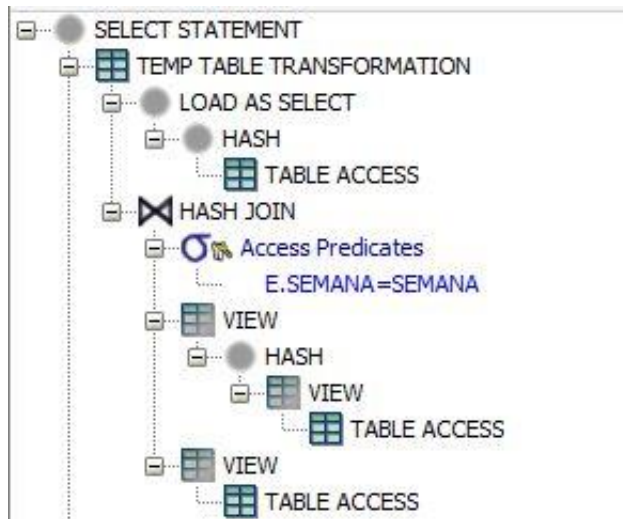


USUARIO	TIPO_IDENTIFICACION	RELACION_USUARIO	OFERTA_ID	CAPACIDAD	PRECI
1 login100005	CEDULA	EGRESADO	11	15	7!

RFC12: Consultar funcionamiento

```
with semana as (  
  SELECT to_number(to_char(to_date(r.fecha_inicio,'MM/DD/YYYY'),'WW')) week_num , MIN(p.dias_reservados) AS personas, p.id as id  
  FROM A_RESERVA r  
  INNER JOIN A_PROPIEDAD p ON (p.id = r.propiedad)  
  group by to_number(to_char(to_date(r.fecha_inicio,'MM/DD/YYYY'),'WW')),p.id  
)  
  
SELECT p.*  
FROM A_PROPIEDAD p  
INNER JOIN semana s ON (s.id = p.id)
```

selecciona todas las propiedades (A_PROPIEDAD) que tienen el máximo número de días reservados (dias_reservados) en una semana específica. La consulta primero calcula el número de semana (week_num) basado en la fecha de inicio de cada reserva (fecha_inicio) utilizando la expresión `to_number(to_char(to_date(r.fecha_inicio,'MM/DD/YYYY'),'WW'))`. Luego une las tablas A_RESERVA y A_PROPIEDAD basándose en el ID de la propiedad (id) para determinar las propiedades con el máximo número de días reservados en cada semana.

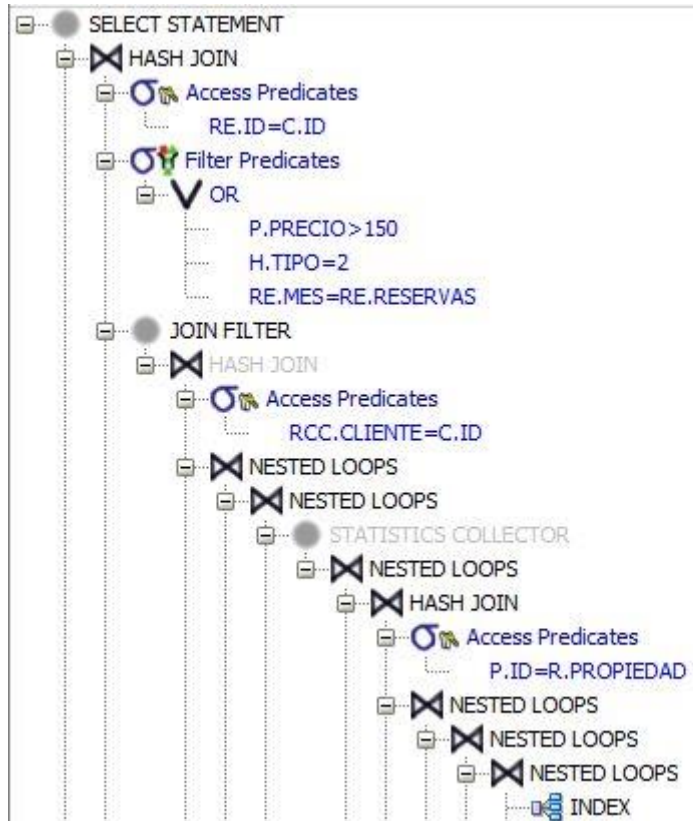


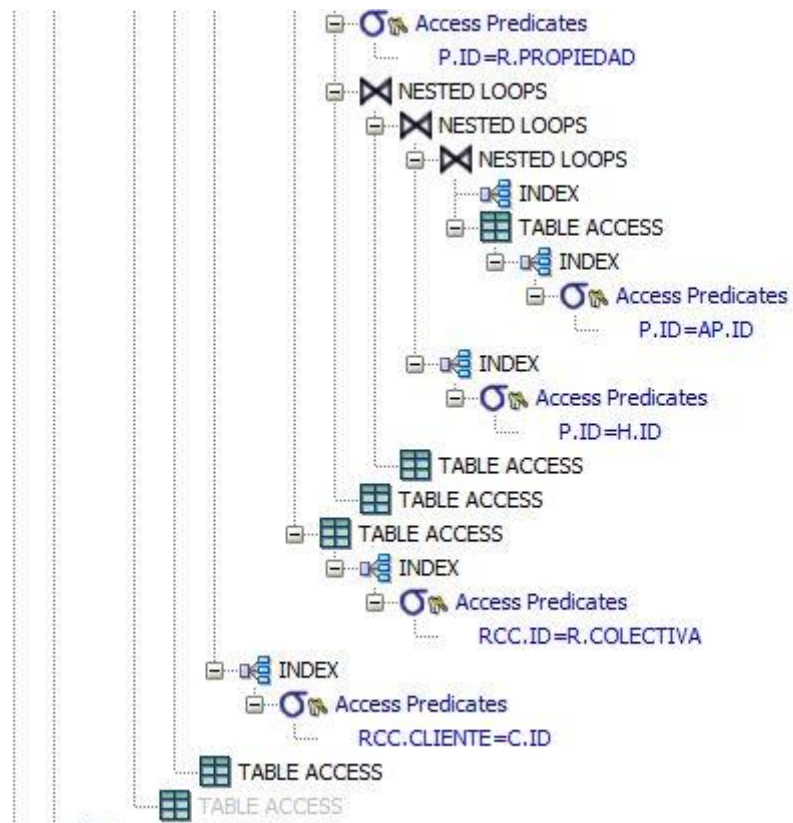
La finalidad de esta consulta es obtener todas las propiedades que están relacionadas con la tabla "semana" según el campo "id". En cuanto al plan de consulta lógico, podemos inferir que Oracle podría seguir los siguientes pasos:

1. Acceder a la tabla "A_RESERVA" y realizar una operación de join con la tabla "A_PROPIEDAD" utilizando la condición de unión "p.id = r.propiedad".
2. Aplicar la función "to_date" para convertir la columna "fecha_inicio" en un formato de fecha adecuado y extraer la semana utilizando "to_char" y "to_number".
3. Agrupar los resultados por semana y propiedad.
4. Calcular el número máximo de personas (días reservados) para cada semana y propiedad.
5. Realizar un join entre la tabla "A_PROPIEDAD" y la subconsulta "semana" utilizando el campo "id" como criterio de unión.
6. Retornar todas las columnas de la tabla "A_PROPIEDAD" en el resultado de la consulta.

RFC13: Consulta buenos clientes

```
with res as ( SELECT count(*) AS mes , c.id as id, count(c.reservas) as reservas
FROM A_CLIENTE c
INNER JOIN A_RESERVA r ON (c.reservas = r.id)
INNER JOIN A_PROPIEDAD p ON (p.id = r.propiedad)
WHERE to_char(r.fecha_inicio,'MM')- to_char((
SELECT r2.fecha_inicio
FROM A_CLIENTE c2
INNER JOIN A_RESERVA r2 ON (c2.reservas = r2.id)
INNER JOIN A_PROPIEDAD p2 ON (p2.id = r2.propiedad)
WHERE c.id = c2.id)) = -1
group by c.id
)
SELECT c.*, p.precio , h.tipo, re.mes
FROM A_CLIENTE c
INNER JOIN A_RESERVA r ON (c.reservas = r.id)
INNER JOIN A_PROPIEDAD p ON (p.id = r.propiedad)
INNER JOIN A_HABITACION h ON (p.id = h.id)
INNER JOIN A_APARTAMENTO ap ON (p.id = ap.id)
INNER JOIN res re ON (re.id = c.id)
WHERE p.precio > 150 OR h.tipo = 3 OR re.mes = re.reservas
```







para seleccionar las ofertas de alojamiento con un precio alto, las habitaciones de un tipo específico y los clientes que cumplen con el criterio de igualdad entre "mes" y "reservas" en la subconsulta "re".

La diferencia fundamental entre la ejecución de consultas delegada al manejador de bases de datos, como Oracle, y una ejecución donde la aplicación trae los datos a memoria principal y resuelve con instrucciones de control (if, while, etc.), radica en el enfoque y los mecanismos utilizados para procesar y optimizar las consultas.

Ejecución de consultas delegada al manejador de bases de datos (Oracle):

Optimización basada en el plan de consulta: El manejador de bases de datos analiza la consulta y genera un plan de consulta optimizado, que es un conjunto de instrucciones de bajo nivel (operadores de acceso, operaciones de join, proyecciones, etc.) para recuperar y manipular los datos de manera eficiente.

Uso de algoritmos y estructuras de datos eficientes: El manejador de bases de datos utiliza algoritmos y estructuras de datos especializados, como índices, estructuras de árbol-B+ y algoritmos de join eficientes, para optimizar el rendimiento de las consultas. Estas técnicas aprovechan la capacidad de procesamiento y la capacidad de almacenamiento del sistema de gestión de bases de datos (SGBD) para realizar operaciones a gran escala de manera eficiente.

Paralelización y optimización de consultas: El manejador de bases de datos puede aprovechar la capacidad de procesamiento paralelo de los servidores para ejecutar consultas de manera eficiente. Además, puede realizar optimizaciones adicionales, como predicción de cardinalidad, ajuste dinámico de consultas y caché de resultados, para mejorar el rendimiento global del sistema.

Ejecución con datos en memoria principal y control manual:

Procesamiento secuencial: En este enfoque, la aplicación debe traer todos los datos relevantes a la memoria principal y realizar el procesamiento de forma secuencial utilizando estructuras de control como if y while. Esto puede implicar una mayor cantidad de operaciones de lectura/escritura en la memoria, lo cual puede ser menos eficiente en comparación con las operaciones optimizadas realizadas por el SGBD.

Falta de optimización automática: La aplicación no se beneficia de las técnicas de optimización automáticas proporcionadas por el SGBD, como la generación de planes de consulta óptimos y el uso de índices. Esto puede resultar en un rendimiento inferior, especialmente cuando se trata de consultas complejas o grandes volúmenes de datos.

Mayor carga de desarrollo y mantenimiento: Al tener que implementar y mantener manualmente la lógica de procesamiento de consultas, la aplicación asume la responsabilidad de optimizar y mantener el rendimiento de las consultas. Esto puede requerir un mayor esfuerzo de

desarrollo y puede ser propenso a errores o subóptimo en comparación con las técnicas de optimización automatizadas proporcionadas por el SMBD.

DISTRIBUCION PARA TODAS LAS CONSULTAS:

Para eso se usan las siguientes suposiciones:

- 170000 de usuarios
- 170000operadores, 170000usuarios
- 20 servicios
- 170000 de reservas: mitad en apartamentos y mitad en habitaciones
- 2000 propiedades: 1000 aptos y 1000 habitaciones
- 7 medios de pago

La distribución en general es la misma para todos los requerimientos, de este modo se puede probar de manera correcta luego con un porcentaje reducido de estos mismos datos para verificar los tiempos de ejecución y los planes de la Query con los índices conocidos, Este enfoque tiene varios beneficios potenciales. Al distribuir los datos de manera uniforme, se puede garantizar que cada requerimiento tenga acceso a una muestra representativa de los datos. Esto es importante porque diferentes consultas pueden tener requisitos y patrones de acceso distintos. Al probar todas las consultas con la misma distribución de datos, se puede detectar de manera más precisa cualquier problema relacionado con los tiempos de ejecución y los planes de consulta.

Una vez que se ha realizado la prueba inicial exhaustiva, se puede proceder a una verificación más detallada utilizando un porcentaje reducido de los datos. Esto implica seleccionar un subconjunto representativo de los datos y realizar pruebas adicionales con ese subconjunto. Al utilizar los índices conocidos, es posible evaluar la efectividad de los planes de consulta y los tiempos de ejecución en un escenario más realista, sin tener que procesar todos los datos.

La ventaja de utilizar un porcentaje reducido de los datos para la verificación es que se puede ahorrar tiempo y recursos, ya que no es necesario procesar todo el conjunto de datos nuevamente. Además, al utilizar índices conocidos, se puede tener una idea más precisa de cómo se comportarán las consultas en el entorno de producción.

Sin embargo, es importante tener en cuenta algunas consideraciones. La distribución uniforme de los datos puede no ser adecuada en todos los casos, especialmente si los datos tienen características específicas que requieren un enfoque de distribución diferente. Además, la selección del

porcentaje reducido de datos para la verificación debe realizarse de manera cuidadosa, para garantizar que la muestra sea representativa y no introduzca sesgos.

Se analizaron las columnas de las tablas para decidir de acuerdo con la selectividad si poner un índice o no, los índices elegidos son los resaltados

Usuario			
Login	TipoID	numeroID	relacionU
0,001	33,33%	0,001	33,33%
Si	No	Si	No

Servicio				
Id	Tipo	Precio	IntervaloDePago	IdPropiedad
0,0005	0,0005	0,0005	0,0005	0,000125
No	No	No	No	Si

Cliente		
Id	TipoID	numeroID
0,0001667	0,255	0,0001
Si	No	Si

Propiedad								
Id	Capacidad	Precio	Tamaño	DiasReserv	Piso	Habilitada	FechaCreacion	Operador
0,000125	0,05	X	X	X	X	0,5	X	0,000166677
Si	No	No	No	No	No	No	X	Si

4 Poblado de tablas

Para la generación e inserción de datos en las respectivas tablas se optó por crear un script en Python que, por medio de la conexión a la base de datos y el uso de la librería faker, se creó una clase Populator cuyos métodos colaboraban a respetar las constraints y se aplicaban las respectivas sentencias INSERT.

Para respetar las constraints de FK se almacenaron los valores necesarios para alimentar al método que necesitaba de los valores asociados a las *foreign keys* para ejecutarse adecuadamente.

5 Estructura física

5.1 Selectividad:

Se toma en cuenta la siguiente cantidad de información para tener en cuenta un análisis detallado del desarrollo y la arquitectura de la estructura física realizada:

- 1 millón de usuarios
- 400.000 operadores, 600.000 usuarios
- 20 servicios
- 1 millón de reservas
- 4 medios de pago

5.2 Índices:

continuación, se detallan los índices recomendados para cada tabla, justificando su selección desde el punto de vista de los requerimientos y considerando el costo de almacenamiento y mantenimiento asociado a los índices.

Tabla A_USUARIO:

Índice primario:

El índice primario ya está creado automáticamente por Oracle. Se utiliza para garantizar la unicidad y la integridad de la clave primaria. No se requiere crear un índice adicional para este requerimiento.

Tabla A_OPERADOR:

Índice primario:

Al igual que en la tabla a_usuario, el índice primario ya está creado automáticamente por Oracle. Proporciona un acceso rápido a los registros por clave primaria.

Índice secundario en la columna "numero_rnt":

Se recomienda crear un índice secundario en esta columna, ya que es utilizada en algunas consultas para buscar operadores por su número de Registro Nacional de Turismo (RNT). El tipo de índice utilizado puede ser un índice B+ debido a su eficiencia en consultas de búsqueda y rangos. Sin embargo, se debe tener en cuenta el costo de almacenamiento adicional asociado a este índice.

Tabla A_CLIENTE:

Índice primario:

El índice primario ya está creado automáticamente por Oracle. Se utiliza para garantizar la unicidad y la integridad de la clave primaria. No se requiere crear un índice adicional para este requerimiento.

Tabla A_OFERTA:

Índice primario:

El índice primario ya está creado automáticamente por Oracle. Proporciona un acceso rápido a los registros por clave primaria.

Índice secundario en la columna "id":

Se recomienda crear un índice secundario en esta columna, ya que se realizan consultas que buscan ofertas por su identificador. El tipo de índice utilizado puede ser un índice B+ debido a su eficiencia en consultas de búsqueda y rangos. Sin embargo, se debe considerar el costo de almacenamiento adicional asociado a este índice.

Tabla A_HABITACION:

Índice primario:

El índice primario ya está creado automáticamente por Oracle. Proporciona un acceso rápido a los registros por clave primaria.

Índice secundario en la columna "id":

Al igual que en la tabla a_oferta, se recomienda crear un índice secundario en esta columna para agilizar las consultas que buscan habitaciones por su identificador. El tipo de índice utilizado puede ser un índice B+ debido a su eficiencia en consultas de búsqueda y rangos. Se debe considerar el costo de almacenamiento adicional asociado a este índice.

Tabla A_APARTAMENTO:

Índice primario:

El índice primario ya está creado automáticamente por Oracle. Proporciona un acceso rápido a los registros por clave primaria. No se requiere crear un índice adicional para este requerimiento.

Tabla A_SERVICIO:

Índice primario:

El índice primario ya está creado automáticamente por Oracle. Se utiliza para garantizar la unicidad y la integridad de la clave primaria. No se requiere crear un índice adicional para este requerimiento.

Tabla A_RESERVACOLECTIVA:

Índice primario: El índice primario ya está creado automáticamente por Oracle.

Índice secundario en la columna "cliente": Se recomienda crear un índice secundario en esta columna, ya que se utilizan consultas que buscan reservas colectivas por el cliente asociado.

Tabla A_RESEVA:

Índice primario:

El índice primario ya está creado automáticamente por Oracle. Proporciona un acceso rápido a los registros por clave primaria.

Índice secundario en la columna "id_oferta":

Se recomienda crear un índice secundario en esta columna, ya que se realizan consultas que buscan reservas por el identificador de la oferta. El tipo de índice utilizado puede ser un índice B+ debido a su eficiencia en consultas de búsqueda y rangos. Sin embargo, se debe tener en cuenta el costo de almacenamiento adicional asociado a este índice.

Tabla A_RESERVAAPARTAMENTO:

Índice primario:

El índice primario ya está creado automáticamente por Oracle.

Tabla A_RESERVAHABITACION:

Índice primario:

El índice primario ya está creado automáticamente por Oracle.

5.2.1

	INDEX_NAME	TABLE_NAME	UNIQUENESS
1	A_APARTAMENTO_PK	A_APARTAMENTO	UNIQUE
2	A_CLIENTE_PK	A_CLIENTE	UNIQUE
3	A_HABITACION_PK	A_HABITACION	UNIQUE
4	A_PROPIEDAD_PK	A_OFERTA	UNIQUE
5	A_OPERADOR_PK	A_OPERADOR	UNIQUE
6	UN_NUMERO_RNT	A_OPERADOR	UNIQUE
7	A_RESERVAI_PK	A_RESERVA	UNIQUE
8	A_RESERVAA_PK	A_RESERVAAPARTAMENTO	UNIQUE
9	A_RESERVA_PK	A_RESERVACOLECTIVA	UNIQUE
10	A_RESERVAH_PK	A_RESERVAHABITACION	UNIQUE
11	A_SERVICIO_PK	A_SERVICIO	UNIQUE
12	A_USUARIO_PK	A_USUARIO	UNIQUE
13	IDX_LOGIN	A_USUARIO	NONUNIQUE
14	CLIENTES_PK	CLIENTES	UNIQUE
15	CONTRATOS_PK	CONTRATOS	UNIQUE
16	INTERESAN_PK	INTERESAN	UNIQUE
17	OFERTAAPARTAMENTO_PK	OFERTAAPARTAMENTO	UNIQUE
18	OFERTAESPORADICA_PK	OFERTAESPORADICA	UNIQUE
19	OFERTAHABITACIONDIARIA_PK	OFERTAHABITACIONDIARIA	UNIQUE
20	OFERTAHABITACIONMENSUAL_PK	OFERTAHABITACIONMENSUAL	UNIQUE
21	OFERTAVIVIENDAUNIVERSITARI_PK	OFERTAVIVIENDAUNIVERSITARIA	UNIQUE
22	PERSONASJURIDICAS_PK	PERSONASJURIDICAS	UNIQUE
23	PERSONASNATURALES_PK	PERSONASNATURALES	UNIQUE
24	PROVEEDORES_PK	PROVEEDORES	UNIQUE
25	RESERVAS_PK	RESERVAS	UNIQUE

5.2.2

Oracle son principalmente índices primarios y algunos índices secundarios en columnas clave o utilizadas frecuentemente en consultas. Estos índices son creados por Oracle para mejorar el rendimiento de las consultas y optimizar la búsqueda de registros en las tablas.

Los índices primarios garantizan la unicidad y la integridad de la clave primaria, lo que permite un acceso rápido a los registros mediante la clave primaria. Estos índices ayudan significativamente al rendimiento de las consultas que buscan registros por su clave primaria.

Los índices secundarios creados en columnas utilizadas frecuentemente en consultas, como "numero_rnt" en la tabla a_operador y "id_oferta" en la tabla a_reserva, mejoran el rendimiento de las consultas que buscan registros basados en estas columnas. Los índices secundarios permiten una búsqueda más eficiente y reducen el tiempo de respuesta de las consultas y en general estos índices ayudan al rendimiento de los requerimientos funcionales al agilizar las consultas y mejorar la eficiencia en la búsqueda de registros. Estos índices optimizan las operaciones de lectura y búsqueda en las tablas, lo que resulta en consultas más rápidas y tiempos de respuesta reducidos. Sin embargo, el costo de almacenamiento adicional y el mantenimiento de los índices es considerable, ya que pueden ocupar espacio en disco y afectar el rendimiento en operaciones de escritura y actualización de datos. Por lo tanto, es necesario evaluar cuidadosamente la necesidad y utilidad de cada índice en función de los requerimientos funcionales y los recursos disponibles.

ELECCION DE ORACLE VS PLANES PROPIOS

Conocimiento interno: El optimizador de consultas de Oracle tiene un conocimiento interno profundo de las estadísticas de la base de datos, como la distribución de datos, el tamaño de las tablas, los índices existentes y las relaciones entre las tablas. Utiliza este conocimiento para generar planes de consulta óptimos que minimizan el tiempo de ejecución y utilizan los recursos de manera eficiente.

Algoritmos sofisticados: El optimizador de consultas de Oracle utiliza algoritmos sofisticados y técnicas avanzadas de optimización de consultas. Puede considerar múltiples métodos de acceso a los datos, evaluar diferentes planes de ejecución y seleccionar el más eficiente en función de las condiciones específicas de la consulta y la estructura de la base de datos.

Estadísticas actualizadas: Oracle mantiene estadísticas actualizadas sobre las tablas y los índices de la base de datos. Estas estadísticas incluyen información sobre la cardinalidad de las columnas, la distribución de los valores y otros datos relevantes. El optimizador utiliza estas estadísticas para estimar el costo de diferentes planes de consulta y elegir el más eficiente en función de los recursos disponibles.

Adaptabilidad: El optimizador de consultas de Oracle puede ajustar sus decisiones en tiempo real en función de la carga de trabajo del sistema y las condiciones cambiantes de la base de datos. Puede reevaluar y ajustar los planes de consulta en tiempo de ejecución para adaptarse a las condiciones actuales y garantizar un rendimiento óptimo.

En resumen, los planes lógicos de consulta generados por el optimizador de consultas de Oracle aprovechan el conocimiento interno, los algoritmos sofisticados, las estadísticas actualizadas y la capacidad de adaptación para producir planes de ejecución eficientes y optimizados. Estos planes suelen superar a los planes creados manualmente por una persona en términos de rendimiento y eficiencia de la consulta.