**“PROYECTO DEL CURSO – ITERACIÓN 4”**

Felipe Gustavo Guzmán Avendaño, Juan Nicolás García Esquibel

Universidad de los Andes, Bogotá, Colombia

{fg.guzman, jn.garciae}@uniandes.edu.co

Fecha de presentación: Mayo 26 de 2020

**Tabla de contenido**

[1 Introducción 1](#_Toc41410179)

[2 Documento 2](#_Toc41410180)

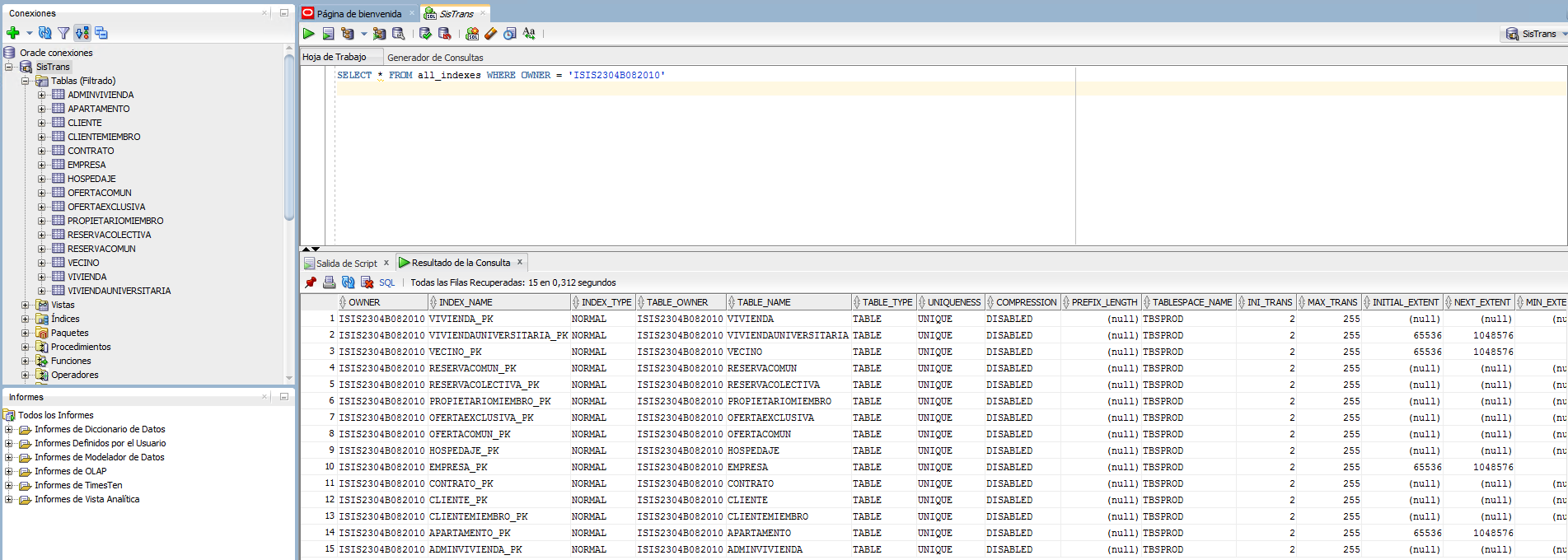
[3 Conclusiones 3](#_Toc41410181)

# Introducción

En el siguiente documento se analiza el caso de AlohAndes para la última iteración, en el cual se analiza el diseño físico de la base de datos y formas de optimizar las consultas para cumplir con los requerimientos ya establecidos.

# Documento

Los índices utilizados son primarios principalmente, esto debido a que la gran cantidad de operaciones que se realizan sobre los datos se basa en el id de las tablas y se almacena como un árbol B+ ya que es el predeterminado por Oracle y es más eficiente que Hash para las búsquedas por rango que son comunes en la realización de este proyecto.



Estos son los índices creados por Oracle que son los primary key de cada tabla, esto se debe a que varias de las consultas radican en el buscar el índice primario en las tablas, al igual que otros requerimientos, sin embargo la llave primaria es más común en la forma que hemos realizado las consultas de requerimientos funcionales.

**Requerimientos funcionales de consulta**

* **RFC 10 Consultar consumo en AlohAndes**

*Se quiere conocer la información de los usuarios que realizaron al menos una reserva de una determinada oferta de alojamiento en un rango de fechas. Los resultados deben ser clasificados según un criterio deseado por quien realiza la consulta. En la clasificación debe ofrecerse la posibilidad de agrupamiento y ordenamiento de las respuestas según los intereses del usuario que consulta como, por ejemplo, por los datos del cliente, por oferta de alojamiento y por tipo de alojamiento.*

**Selección de índices:** Se tomó la decisión de implementar un índice sobre fechaInicio, debido a que esto mostraría un rango de fechas iniciales en las que se hacían las reservas, de tal manera que es más fácil encontrar los usuarios dada la fecha parámetro y como en este requerimiento importan más las fechas que los usuarios en sí, por eso se seleccionó este índice.

**Sentencia de creación del índice:**

CREATE UNIQUE INDEX IN\_FECHA ON RESERVACOMUN (fechaInicio)

**Sentencia SQL Utilizada:**

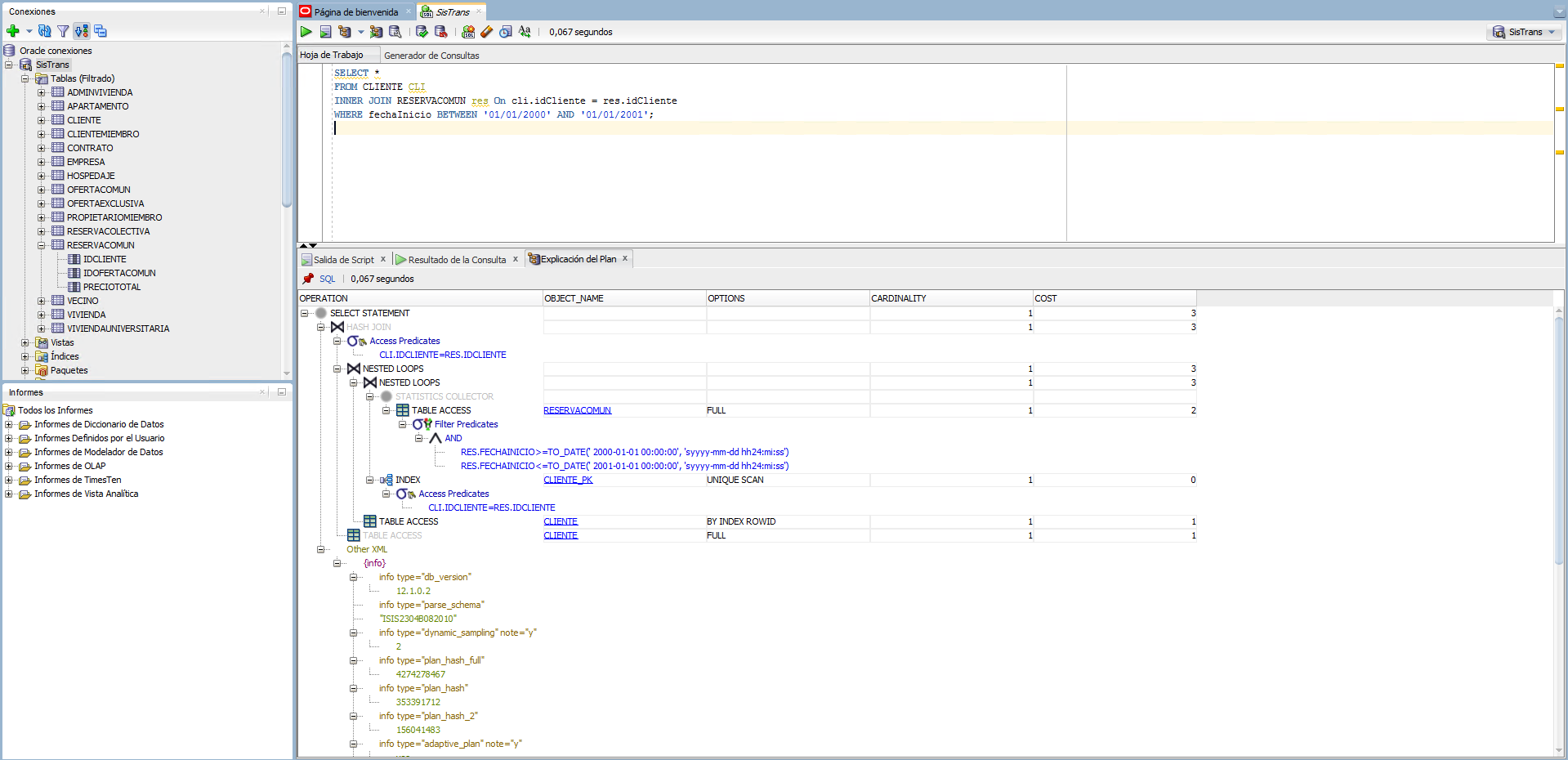
SELECT \*

FROM CLIENTE CLI

INNER JOIN RESERVACOMUN res On cli.idCliente = res.idCliente

WHERE fechaInicio BETWEEN ' ' AND ' '

**Plan de consulta**

****

* **RFC 11 Consultar consumo en AlohAndes**

*Se quiere conocer la información de los usuarios QUE NO realizaron al menos una reserva de una determinada oferta de alojamiento en un rango de fechas. En la clasificación debe ofrecerse la posibilidad de agrupamiento y ordenamiento de las respuestas según los intereses del usuario que consulta como, por ejemplo, por los datos del cliente, por oferta de alojamiento y por tipo de alojamiento.*

**Selección de índices:** Como la consultar es bastante similar a la anterior y se hace según los mismos parámetros, se usa exactamente el mismo índice creado previamente.

**Sentencia SQL Utilizada:**

SELECT cli.idCliente, res.idOferta AS ofe

FROM CLIENTE cli

LEFT JOIN RESERVACOMUN res On cli.idCliente = res.idCliente

WHERE fechaInicio BETWEEN '' AND '' AND ofe IS NULL

* **RFC 12 Consultar funcionamiento**

*Muestra, para cada semana del año, la oferta de alojamiento con más ocupación, la oferta de alojamiento con menos ocupación, los operadores más solicitados y los operadores menos solicitados. Las respuestas deben ser sustentadas por el detalle de las ofertas de alojamiento y operadores correspondientes. Esta operación es realizada el gerente general de AlohAndes.*

**Selección de índices:** Puede ser sobre oferta comun

**Sentencia SQL utilizada:**

SELECT emp.idOperador AS idOp, emp.nombre AS nombreOperador

FROM EMPRESA emp

INNER JOIN (Select oc.idOfertaComun AS idOfe, COUNT(\*) AS numReservas

FROM OfertaComun oc

INNER JOIN ReservaComun rc ON oc.idOfertaComun = rc.idOfertaComun

GROUP BY oc.idOfertaComun

ORDER BY numReservas DESC) AUX ON emp.idOperador = AUX.idOfe

UNION

SELECT vec.idOperador, vec.nombre

FROM VECINO vec

INNER JOIN AUX ON vec.idOperador = AUX.idOfe

* **RFC 13 Consultar los buenos clientes**

*Los buenos clientes son de tres tipos: aquellos que hacen reservas en AlohAndes al menos una vez al mes, aquellos que siempre reservan alojamientos costosos (Entiéndase costoso, por ejemplo, como mayor a USD 150 por noche) y aquellos que siempre reservan suites. Esta consulta retorna toda la información de dichos clientes, incluyendo la que justifica su calificación como buenos clientes. Esta operación es realizada únicamente por el gerente general de AlohAndes.*

**Selección de índices:** idCliente en reservaComun.

**Sentencia SQL utilizada:**

SELECT cli.idCliente, COUNT(\*) AS numRes

FROM CLIENTE cli

INNER JOIN RESERVACOMUN rc ON cli.idCliente = rc.idCliente

WHERE numRes/12 == 1 OR rc.preciototal >= 150

GROUP BY cli.idCliente

# Conclusiones

Al final de la realización de este proyecto se logró una mejor comprensión sobre la aplicación en Java y la implementación de las consultas sobre la base de datos para conseguir los datos necesarios según sea la consulta. Se consiguió una implementación adecuada de la nueva clase así como las modificaciones necesarias a las demás tablas para una correcta funcionalidad.

La implementación de transaccionalidad es eficiente, se maneja un nivel de aislamiento de read committed para las transacciones ya que un serializable involucraría un peso grande en el sistema de datos teniendo en cuenta que se deben manejar varios usuarios en simultaneo, tanto clientes como operadores.

Para los nuevos requerimientos de consulta, se hace una búsqueda razonable sobre la base de datos para luego, con los datos obtenidos, seleccionar desde java los necesarios según se requiera en la consulta y hacerla más específica a lo que se busca.

Para concluir, se utilizaron correctamente los temas aprendidos durante el curso con respecto a transaccionalidad pues se identificaron los posibles abrazos mortales y los pasos para evitarlos, así como una mayor comprensión de la conexión entre Java y SQL Developer para mandar y recibir datos.