ITERACIÓN 4 – DISEÑO FÍSICO Y OPTIMIZACIÓN DE CONSULTAS

Federico Gadea (f.gadea)

8 de dic. de 20

ANALISIS:

En el modelo del mundo se actualizó todas las clases que tenían un atributo de tipo Date, se cambió el tipo de todos estos atributos a tipo String.

DISEÑO DE LA APLICACIÓN:

* IMPACTO: Los nuevos requerimientos exigieron un leve cambio en los tipos de datos que se estaban manejando. Debido a que tenía un error para el que no encontré solución, tuve que realizar un FORK del ejemplo del proyecto propuesto por la clase y modificarlo a conveniencia. La consistencia del proyecto es la misma solo que evito la modificación de archivos que no tuve que modificar en la iteración anterior.
* DISEÑO FÍSICO:

Oracle para la búsqueda de los requerimientos RFC10 y RFC11 seguramente utilizaría un Arbol b+ primario que tenga la referencia hacia las fechas de la visita. Se utiliza un árbol primario ya que un árbol funciona correctamente para extraer la información entre un rango.

*Nota: no logré correr la aplicación sin embargo no quiero perder todos los puntos de esta sección.*

* Requerimientos de consulta y su análisis:

RFC10 - CONSULTAR VISITAS EN AFORO-CCANDES:

Text

Description automatically generated

El requerimiento recibe por parámetro el id del local que se quiere analizar, un rango de fechas (fechaInicio y fechaFin) y un String Ordenar que tiene como referencia al atributo por el cual se requiera ordenar el resultado.

La función BETWEEN permite seleccionar un rango, en este caso de fechas.

Es importante darnos cuenta de que el ‘peso’ de la búsqueda depende del rango de fechas que se quieran analizar. En este caso, si el rango de fechas es suficientemente grande podemos suponer que es mejor revisar todo el archivo con una lectura rápida que utilizar algún índice. Por esta razón la decisión de Oracle a la hora de seleccionar su ‘algoritmo’ debe de tomar en cuenta principalmente las fechas.

RFC11- CONSULTAR VISITAS EN AFORO-CCANDES – RFC10-V2

Text

Description automatically generated

El código utilizado es el mismo con respecto a RFC10 con la excepción que se utiliza NOT IN envés de IN para el análisis de un local en un determinado rango de fechas. El NOT IN me permite consultar los usuarios que no estuvieron en un local en un determinado rango. Es importante recalcar que para esta función posiblemente se utilice una lectura rápida porque generalmente la cantidad de personas en un solo local será mucho menor que las personas por fuera, entonces seguramente es más rápido revisar todos los datos.

En esta función pasa lo contrario a la anterior, y a simple vista parece ser que utilizar un índice no es efectivo.

CONSTRUCCION DE LA APLICACIÓN, EJECUCION DE PRUEBAS Y ANÁLISIS DE RESULTADOS:

* DISEÑO DEL ESCENARIO DE PRUEBAS DE EFICIENCIA:
* Diseño de datos:

Se utilizó la librería FAKER para generar datos que tengan sentido en la aplicación. Esta librería genera predeterminadamente información que cumpla con los parámetros de aleatoriedad y vayan de acuerdo con lo exigido.

Ejemplo:

public void generarVisitantes () {

for (int i = 0; i < 100000; i++){

String name = faker.name().fullName();

String correo = faker.internet().emailAddress();

String numTelefono = faker.number().digits(10);

String name2 = faker.name().fullName();

String numTelefono2 = faker.phoneNumber().cellPhone();

Double temperatura = faker.number().randomDouble(1, 35, 40);

pp.adicionarVisitante(name, tipo, Integer.parseInt(numTelefono), correo, name2, Integer.parseInt(numTelefono2), estado, temperatura);}}

Como se aprecia en el ejemplo, faker proporciona datos que cumplen con los parámetros buscados. De esta forma con un loop genero cien mil visitantes cada uno con un nombre, correo, etc aleatorios.

En el documento se exigen al menos 1 millón de datos, por lo tanto la suma de los datos visitantes+ visitas+ lectores+ carnet+… dan en total 1 millón. Visita, que es donde se deberían presentar más datos, tiene un total de 820955 datos.

Dato interesante con librería faker:

Durante la creación de una visita, se asocia a un id random de usuario entre 1 y cien mil para que los usuarios tengan varias o ninguna visita y así tener datos que si posean variedad.

La función utilizada:

int id\_visitante = faker.number().numberBetween(1,100000);

*No utilicé SQL LOADER (Recomendad) porque me pareció más fácil añadir los datos de esta manera.*

Para añadir los datos a las tablas se utiliza la clase persistenciaAforo y el método adicionar*Tipodedato*() pasando por parámetro los datos aleatorios correspondientes.

public void generarTodo() {

generarHorario();

generarCC();

generarLector();

generarVisitantes();

generarCarnet();

generarEspacio();

generarLocal();

generarLectorCC();

generarLectorEspacio();

generarVisita();

}

Para que no haya conflicto, los datos se tienen que generar en el siguiente orden.