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:

```
public List<Visitante> RFC10(PersistenceManager pm, Long id_local, String fechaInicio, String fechaFin, String ordenar)
{
    String sql = "SELECT *";
    sql +=" FROM " +pp.darVisitantes();
    sql+=" WHERE ";
    sql+="id IN";
    sql+="(SELECT id_visitante FROM" + pp.darSeqVisita()+ "WHERE FechaHoraEntrada BETWEEN\n" + fechaInicio +"AND" + fechaFin;
    sql+="AND id_lector IN";
    sql+="(SELECT id FROM"+pp.darLectoresEspacio();
    sql+="WHERE idEspacio="+id_local+"));";

    if(ordenar !="" && ordenar !=null)
    {
        sql+="ORDER BY"+ ordenar +";";
    }

    Query q = pm.newQuery(SQL, sql);
    return (List<Visitante>) q.executeList();
}
```

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. Sin embargo,un índice Arbol B+ sería el ideal para realziar la consulta.

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

```
public List<Visitante> RFC11(PersistenceManager pm, Long id_local, String fechaInicio, String fechaFin, String ordenar)
{
    String sql = "SELECT *";
    sql +=" FROM " +pp.darVisitantes();
    sql+=" WHERE ";
    sql+="id IN";
    sql+="(SELECT id_visitante FROM" + pp.darSeqVisita()+ "WHERE FechaHoraEntrada BETWEEN\n" + fechaInicio +"AND" + fechaFin;
    sql+="AND id_lector NOT IN";
    sql+="(SELECT id FROM"+pp.darLectoresEspacio();
    sql+="WHERE idEspacio="+id_local+"));";

    if(ordenar !="" && ordenar !=null)
    {
        sql+="ORDER BY"+ ordenar +";";
    }
    Query q = pm.newQuery(SQL, sql);
    return (List<Visitante>) q.executeList();
}
```

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.

RFC12 - CONSULTAR FUNCIONAMIENTO

```
List<Establecimiento> rf12(PersistenceManager pm){
ring sql1="";
l1+="CREATE TABLE LOCALES_VISITADOS(\n" +
      "NOMBRE VARCHAR2(255 BYTE) NOT NULL,\n" +
      "TIPO VARCHAR2(255 BYTE) NOT NULL,\n" +
      "CONTADOR NUMBER NOT NULL,\n" +
      "FECHA VARCHAR2(255 BYTE) NOT NULL\n" +
      ");\n" +
      "DECLARE\n" +
      "V_ITERATOR NUMBER;\n" +
      "V_DATE VARCHAR2(255 BYTE);\n" +
      "BEGIN\n" +
      "V_ITERATOR:= 0;\n" +
      "V_DATE:= TO_DATE('01/01/2020', 'DD/MM/YYYY');\n" +
      "LOOP\n" +
      "V_ITERATOR:= V_ITERATOR+7;\n" +
      "INSERT INTO LOCALES_VISITADOS\n" +
      "SELECT ESTABLECIMIENTO.nombre, ESTABLECIMIENTO.tipo_establecimiento as Tipo,contadorVisitas,V_DATE\n" +
      "FROM(SELECT IDESPACIO as IDESPACIO, COUNT( ID_VISITANTE)as contadorVisitas\n" +
      "     FROM" + pp.darSeqVisita()+
      "     WHERE VISITA.FECHAENTRADA BETWEEN V_DATE AND V_DATE+V_ITERATOR\n" +
      "     GROUP BY IDESPACIO\n" +
      "     HAVING COUNT( ID_VISITANTE)>0)\n" +
      "     INNER JOIN LOCAL_COMERCIAL ON IDESPACIO=LOCAL_COMERCIAL.idespacio\n" +
      "     WHERE ROWNUM<=1\n" +
      "     ORDER BY contadorVisitas DESC;\n" +
      "V_DATE:= V_DATE+V_ITERATOR+1;\n" +
      "     EXIT WHEN V_ITERATOR=364;\n" +
      "END LOOP;\n" +
      "END;\n" +
      "SELECT * FROM LOCALES_VISITADOS;\n" +
      "CREATE TABLE LOCALES_MENOS_VISITADOS(\n" +
      "NOMBRE VARCHAR2(255 BYTE) NOT NULL,\n" +
      "TIPO VARCHAR2(255 BYTE) NOT NULL,\n" +
      "CONTADOR NUMBER NOT NULL,\n" +
      "FECHA VARCHAR2(255 BYTE) NOT NULL\n" +
      ");\n" +
      "DECLARE\n" +
      "V_ITERATOR NUMBER;\n" +
      "V_DATE VARCHAR2(255 BYTE);\n" +
      "BEGIN\n" +
      "V_ITERATOR:= 0;\n" +
      "V_DATE:= TO_DATE('01/01/2020', 'DD/MM/YYYY');\n" +
      "LOOP\n" +
      "V_ITERATOR:= V_ITERATOR+7;\n" +
      "INSERT INTO LOCALES_MENOS_VISITADOS\n" +
      "SELECT ESTABLECIMIENTO.nombre, ESTABLECIMIENTO.tipo as Tipo,contadorVisitas,V_DATE\n" +
      "FROM(SELECT IDESPACIO as IDESPACIO, COUNT( ID_VISITANTE)as contadorVisitas\n" +
      "     FROM" + pp.darSeqVisita()+
      "     WHERE VISITA.FECHAENTRADA BETWEEN V_DATE AND V_DATE+V_ITERATOR\n" +
      "     GROUP BY IDESPACIO\n" +
      "     HAVING COUNT( ID_VISITANTE)>0)\n" +
      "     INNER JOIN LOCAL_COMERCIAL ON IDESPACIO=LOCAL_COMERCIAL.idespacio\n" +
      "     ORDER BY contadorVisitas ASC, V_DATE ASC;\n" +
      "V_DATE:= V_DATE+V_ITERATOR+1;\n" +
      "     EXIT WHEN V_ITERATOR=364;\n" +
      "END LOOP;\n" +
      "END;\n" +
      "CREATE TABLE LOCALES_MAS_VISITADOS(\n" +
      "NOMBRE VARCHAR2(255 BYTE) NOT NULL,\n" +
      "TIPO VARCHAR2(255 BYTE) NOT NULL,\n" +
      "CONTADOR NUMBER NOT NULL,\n" +
      "FECHA DATE NOT NULL\n" +
      ");\n" +
      "DECLARE\n" +
      "V_ITERATOR NUMBER;\n" +
      "V_DATE VARCHAR2(255 BYTE);\n" +
      "BEGIN\n" +
      "V_ITERATOR:= 0;\n" +
      "V_DATE:= TO_DATE('01/01/2020', 'DD/MM/YYYY');\n" +
      "LOOP\n" +
      "V_ITERATOR:= V_ITERATOR+7;\n" +
      "INSERT INTO LOCALES_MAS_VISITADOS\n" +
      "SELECT * FROM LOCALES_MENOS_VISITADOS WHERE CONTADOR =(SELECT MIN(contador) FROM LOCALES_MENOS_VISITADOS WHERE FECHA BETWEEN V_DATE AND V_DATE+V_ITERATOR);\n"
      "V_DATE:= V_DATE+V_ITERATOR+1;\n" +
      "     EXIT WHEN V_ITERATOR=364;\n" +
      "END LOOP;\n" +
      "END;\n" +
      "SELECT DISTINCT * FROM LOCALES_MAS_VISITADOS;";
ery q = pm.newQuery(SQL,sql1);
turn (List<Establecimiento>) q.executeList();
```

El método básicamente crea 3 tablas donde se van a guardar los resultados y los muestra. En teoría lo mejor es utilizar una lectura rápida esto se debe a que se requiere revisar todas las tablas por lo tanto un índice no tiene sentido.

RFC13 - CONSULTAR LOS BUENOS VISITANTES

```
public List<Visitante> buenosVisitantes(PersistenceManager pm)
{
    String sql = "(SELECT A_VISITANTE.ID as id, A_VISITANTE.NOMBRE as nombre, A_VISITANTE.TELEFONO as telefono, "
            + "A_VISITANTE.NOMCONTACTO as contacto, A_VISITANTE.NUMCONTACTO as contactoTelefono, A_VISITANTE.CORREO as correo";
    sql+="FROM(SELECT c, COUNT(DISTINCT mes)";
    sql+="ROM(SELECT EXTRACT(MONTH FROM CAST(FECHAENTRADA as DATE)) as mes,ID_VISITANTE as c";
    sql+="FROM " + pp.darSeqVisita();
    sql+="GROUP BY EXTRACT(MONTH FROM CAST(FECHAENTRADA as DATE)),ID_VISITANTE)";
    sql+="GROUP BY c";
    sql+="HAVING COUNT (DISTINCT mes)='12')";
    sql+="INNER JOIN A_VISITANTE ON A_VISITANTE.ID=VISITA.id_visitante) UNION";
    sql+="SELECT A_VISITANTE.ID as id, A_VISITANTE.NOMBRE as nombre, A_VISITANTE.TELEFONO as telefono,"
            + " A_VISITANTE.NOMCONTACTO as contacto, A_VISITANTE.NUMCONTACTO as contactoTelefono, A_VISITANTE.CORREO as correo";
    sql+="FROM (SELECT dia, c, COUNT (DISTINCT Establecimiento) as espaciosVisitados";
    sql+="FROM(SELECT EXTRACT(DAY FROM CAST(FECHAENTRADA as DATE)) as dia, ID_VISITANTE as c, IDESTABLECIMIENTO as Establecimiento, TIPO ";
    sql+="FROM " +  pp.darSeqVisita();
    sql+="GROUP BY FECHAENTRADA, ID_VISITANTE, IDESTABLECIMIENTO, TIPO, EXTRACT(DAY FROM CAST(FECHAENTRADA as DATE))";
    sql+="HAVING EXTRACT(MONTH FROM CAST(FECHAENTRADA as DATE))='12' AND EXTRACT(YEAR FROM CAST(FECHAENTRADA as DATE))='2020' AND TIPO='Entrada')";
    sql+="GROUP BY dia, c";
    sql+="HAVING COUNT (DISTINCT Establecimiento)>=4)";
    sql+="INNER JOIN A_VISITANTE ON A_VISITANTE.ID=VISITA.id_visitante) UNION(";
    sql+="SELECT A_VISITANTE.ID as id, A_VISITANTE.NOMBRE as nombre, A_VISITANTE.TELEFONO as telefono, "
            + "A_VISITANTE.NOMCONTACTO as contacto, A_VISITANTE.NUMCONTACTO as contactoTelefono, A_VISITANTE.CORREO as correo";
    sql+="FROM(SELECT c, COUNT (DISTINCT tipo) tipoEstablecimiento";
    sql+="FROM(SELECT dia, c, Establecimiento, TIPO as tipo";
    sql+="FROM(SELECT EXTRACT(DAY FROM CAST(FECHAENTRADA as DATE)) as dia, ID_VISITANTE as c, IDESTABLECIMIENTO as Establecimiento, TIPO";
    sql+="FROM " +  pp.darSeqVisita();
    sql+="GROUP BY FECHAENTRADA, ID_VISITANTE, IDESTABLECIMIENTO, EXTRACT(DAY FROM CAST(FECHAENTRADA as DATE))";
    sql+="HAVING EXTRACT(MONTH FROM CAST(FECHAENTRADA as DATE))='12' AND EXTRACT(YEAR FROM CAST(FECHAENTRADA as DATE))='2020')";
    sql+="INNER JOIN ESTABLECIMIENTO ON establecimiento.idespacio=Establecimiento)";
    sql+="GROUP BY c";
    sql+="HAVING COUNT (DISTINCT tipo)>=2)";
    sql+="INNER JOIN A_VISITANTE ON A_VISITANTE.ID=VISITA.ID_visitante);";

    Query q = pm.newQuery(SQL, sql);
    return (List<Visitante>) q.executeList();
}
```

Esta función está "dividida" en tres partes para cumplir con los parámetros que se requieren. En este caso por los filtros que son de rango mi recomendación es utilizar un índice Arbol B+ primario que acorte los rangos de búsqueda. Podría ser recomendado que se realice por fecha.

R

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);

    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 830955 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);`

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.

```
public void LoadData(PersistenceManager pm)
{
    String sql = "";
    File fr = new File("Iteracion4_f.gadea_/data/datos.txt");
    try {
        Scanner myReader = new Scanner(fr);
        while(myReader.hasNextLine())
        {
            sql += myReader.nextLine();
            if(myReader.nextLine()==";")
            {
                Query q = pm.newQuery(SQL, sql);
                q.executeList();
                sql="";
            }
        }

    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

ANALISIS:

Según entiendo las operaciones en Oracle tienen un nivel más en complejidad algorítmica (no me refiero a eficiencia) sino a la construcción en sí de la operación. Un if es lo mismo que un Where. Un Join sería algo que junta y asocia dos tablas, podría ser comparable a un arreglo que apunte a otro arreglo.
Las operaciones en Oracle al trabajar con manejo de base de datos generalmente utilizan datos que no están en memoria, mientras que en JAVA sí por lo que el tiempo de operación en teoría es menor (para JAVA) y esto se debe a que buscar en disco es una operación relativamente tardada.