Trabajo 3 BD2-2021 - Bases de datos de documentos y objeto-relacionales

En este trabajo, usamos una base de datos objeto-relacional en OracleDB y otra base de datos de documentos en MongoDB.
Realizamos diferentes operaciones entre ambas bases de datos, por ejemplo, extrayendo de una e insertando en la otra.

Conexiones y Drivers

Oracle JDBC

Para usar la base de datos de Oracle en Java, usamos el driver JDBC de Oracle y seguimos la documentación para establecer una conexión y realizar declaraciones.

MongoDB Driver

Para usar la base de datos de MongoDB, usamos el driver mongodb-driver-sync. Intentamos usar codecs para mappear de consultas a POJOs (Plain old Java Object), pero no fue posible, por lo que lo hacemos manualmente.

Consultas

Generar estadísticas desde OracleDB y guardar en MongoDB

Para generar las estadísticas iniciales desde OracleDB, usamos la siguiente consulta:

```
SELECT *
FROM (
         SELECT cc,
                e.miciu.nom       AS                            ciudad,
                e.miciu.midep.nom AS                            departamento,
                SUM(v.miprod.precio_unitario * v.nro_unidades) total,
                rank() over (partition by e.miciu.nom order by SUM(v.miprod.precio_unitario * v
         FROM empleado e,
              TABLE (e.ventas) v
         GROUP BY e.cc, e.miciu.midep.nom, e.miciu.nom
         ORDER BY total desc
     )
WHERE rank = 1;
```

Esta consulta rankea `rank()` a los empleados basado en el total de ventas `SUM(v.miprod.precio_unitario * v.nro_unidades)` y partido por los nombres de cada ciudad, obteniendo un resultado como este:

CC	CIUDAD	DEPARTAMENTO	TOTAL	RANK
10	Bogotá	Cundinamarca	500000	1
190	Medellín	Antiqouia	500000	1
5	Medellín	Antiqouia	250000	2

Después, filtramos a los mejores de cada ciudad `rank = 1` .

En Java, obtenemos este resultado e iteramos por cada fila para generar una lista de `src/main/java/org/unalmed/models/EstadisticaDepartamento` . Como tuvimos problemas con los codecs de Mongo, no pudimos insertar esta lista de objetos directamente, por lo que tuvimos que crear una lista de documentos a partir de estos y después invocar el método `insertMany` de `MongoCollection` .

Visualizar estadísticas

Para generar las estadísticas globales, usamos el *Aggregation Pipeline* de MongoDB y las separamos en estadísticas de regiones y de vendedores:

Para generar las estadísticas de los vendedores, generamos la siguiente "tubería":

```
// Vendedores
[
  {
    '$unwind': {
      'path': '$misventas'
    }
  }, {
    '$sort': {
      'misventas.total_vendedor': -1
    }
  }, {
    '$group': {
      '_id': '',
      'mejor_vendedor': {
        '$first': '$misventas'
      },
      'peor_vendedor': {
        '$last': '$misventas'
      }
    }
  }
]
```

- `$unwind` : Deconstruye un item de un array a partir de los documentos de entrada para generar un documento para cada elemento, por lo que genera un documento por cada item en `$misventas` .
- `$sort` : Ordena todos los documentos de entrada, descendentemente en este caso.
- `$group` : Agrupa los documentos de entrada por la expresión _id especificada (en este caso es vacía, por lo que agrupa todos los documentos) y para cada agrupación distinta, genera un documento. Los documentos de salida también pueden contener campos calculados que contienen los valores de alguna expresión de acumulador (en nuestro caso, `mejor_vendedor` es el primer item (`$first`) de la lista `$misvetas` , ordenada descendentemente en el paso anterior y `peor_vendedor` es el último item (`$last`)).

Que al traducirlo a Java y refactorizando, tenemos:

```
Bson unwind = unwind("$misventas");
Bson sort = sort(descending("misventas.total_vendedor"));
BsonField first = first("mejor_vendedor", "$misventas");
BsonField last = last("peor_vendedor", "$misventas");
Bson group = group("", first, last);

AggregateIterable<Document> vendedor = this.estadisticasCollection.aggregate(
        Arrays.asList(unwind, sort, group));
```

Para las estadísticas de las regiones y las estadísticas de cada departamento, usamos otros pipelines que son muy complejos para este README. Las versión de Java de estas tuberías se encuentran en `src/main/java/org/unalmed/daos/EstadisticaDAO @ getEstadisticasRegion (línea 302)` y `src/main/java/org/unalmed/daos/EstadisticaDAO @ getEstadisticasDepartamentos (línea 243)` respectivamente.

Generar históricos y vaciar arreglos de ventas en OracleDB

Para generar los históricos, ejecutamos la siguiente sentencia:

```
MERGE INTO HISTORICOVENTAS h
        USING ((SELECT cc,
                SUM(v.miprod.precio_unitario * v.nro_unidades) totalacumuladoventas
                FROM empleado e,
                TABLE (e.ventas) v
                GROUP BY e.cc)) t
        ON (t.CC = h.CC)
    WHEN MATCHED THEN
        UPDATE SET h.TOTALACUMULADOVENTAS = h.TOTALACUMULADOVENTAS + t.totalacumuladoventas
    WHEN NOT MATCHED THEN
        INSERT (cc, totalacumuladoventas)
        VALUES (t.cc, t.totalacumuladoventas);
```

La cual usa `MERGE INTO`, el cual usando la tabla generada del `SELECT` (query modificada del punto 1), cuando encuentra que la cédula ya tiene un histórico, suma al campo lo que ya tenía y lo actualiza. Si no encuentra la cédula, inserta un nuevo registro.

Para vaciar los arreglos de `misventas` en la tabla `EMPLEADO`, ejecutamos la siguiente sentencia:

```
UPDATE EMPLEADO e SET e.VENTAS = NULL WHERE e.cc = e.cc;
```