AMPLIACION DE BASE DE DATOS
Examen Final Convocatoria Ordinaria 2021. --- una SOLUCION --

EJERCICIO 1 (3 puntos)

Tenemos creada en MongoDB una colección llamada `MiOcio`, dentro de una base de datos llamada `miDB`, cuyos documentos (uno por cada actividad de ocio) tienen los siguientes campos:

- *TipoOcio*: EJ: viajes, música, esquí, cine, etc.
- *Empresa*: quien oferta ese tipoOcio. Ej: Warner, Sony, etc.
- *Elemento*: Nombre de la actividad concreta. EJ: viajeCanarias, conciertoU3, JuegosTronos, etc.
- *Nivel:* de popularidad, número estrellas que tiene el elemento (entre 0 y 10)
- *Coste*: EJ: precio del viajeCanarias, precio del conciertoU3, etc.
- *Propiedades*: array de pares (nombre, valor) de características particulares del elemento

La colección `MiOcio` se asume que ya tiene documentos

SE PIDE: Escribe las instrucciones en MongoDB necesarias para los siguientes apartados.

a) Usando **`aggregate`** y solo de las actividades más populares (Nivel > = 9), queremos obtener *para cada* TipoOcio y Empresa estos valores: el TipoOcio, la Empresa, el máximo de los costes de sus actividades, y cuántas actividades hay que cumplen lo indicado. Queremos esos valores solo para los TipoOcio que tengan más de 3 actividades. Clasifica el resultado de modo descendente por TipoOcio y Empresa.

```
db.MiOcio.aggregate([
   { $match: { "Nivel":{$gte:9} } },
   { $group: { _id: { ElTipo: "$TipoOcio", LaEmpresa: "$Empresa" },
               maxCoste: { $max: "$Coste" }, totdocs: { $sum: 1 } } },
   { $match: { "totdocs":{$gte:3} } } ,
   {$sort: { _id: -1}}
   ])
```

b) Usando solo una instrucción **`update`** (sin funciones que recorran la colección). Queremos actualizar el 1º documento que cumpla las condiciones siguientes: TipoOcio: "ciclismo", Empresa: "miBici", Elemento: "Derbi" y Propiedades debe tener el par : "(longitud , 200km)". Además, no queremos que otro proceso *pueda escribir en la colección* hasta terminar esta operación. Los *cambios* a realizar cuando se cumplen esas condiciones son: Nivel = 9, Coste = 5, y un atributo nuevo, ConApoyo: sí.
Si no existe ningún documento que cumpla las condiciones, se insertará el documento que incluya todos estos datos.

```
db.MiOcio.update({$and: [{TipoOcio: " ciclismo"},
                         {Empresa: " miBici"},
                         {Elemento: " Derbi"},
                         {Propiedades: {$elemMatch: {("longitud" : "200km")}}}
                        ],
                          $isolated:1}, /* otro proceso no escribe hasta terminar */
                  $set: { Nivel: 9, Coste: 5,  ConApoyo: "sí" },
                  { upsert: true }, false )  //  literal "upsert" solo para resaltar
                                    /* si no existe crea este doc con upsert*/
```

c) Usando solo una instrucción **`find`** con la función **`forEach`** que recorra la colección para actualizar. Queremos actualizar solo aquellas actividades cuyo TipoOcio es alguno de estos: "ciclismo", "rafting", "piragua", "lectura". La actualización consiste en bajar el coste proporcionalmente, de aquellas actividades que tienen el nivel bajo (<5):
- Si el coste es menor de 500, lo actualizamos usando esta fórmula (10 - (5 – nivel) /10) * coste. Además, añadimos el atributo: Estado = "ganga".
- En caso contrario actualizamos coste usando: (8 - (5 – nivel) /10) * coste; Además, añadimos el atributo: Estado = "dudoso".
-> Después, además de actualizarla, imprime cada actividad.

```
db.MiOcio.find({TipoOcio: {$and [ { Nivel: {$lt : 5}
                  {$in: ["ciclismo", “rafting”,“piragua”, “lectura”] }] }).forEach(
  function (myDoc) {
     if (myDoc.Coste <  500 ){
         myDoc.Coste = ((10 - (5 - myDoc.Nivel)) / 10) * myDoc.Coste;
         myDoc.Estado = “ganga”;
        }
      else {
          myDoc.Coste = ((8 -(5 - myDoc.Nivel)) /10) * myDoc.Coste;
         myDoc.Estado = “dudoso”;
       }
     print(myDoc);   /* para ver lo que modifica */
     db.Ocio.save(myDoc);  /* también se puede con update */
  });
```

EJERCICIO 2 (6 puntos)

Dado el siguiente esquema relacional (las tablas están creadas y con datos)

Cliente: CL(DNI, NombreC,Dirección)
Invierte: I(DNI, NombreE,Cantidad,Tipo)
Compras_XXXXX: CO(NumT, NumFac, Fecha, Tienda, Importe)

→ Hay una tabla de compras para cada cliente: en el nombre de la tabla, XXXXX es su DNI

a) Hacer un procedimiento en PL/SQL llamado *EJ2* que incluya los siguientes pasos (usa un cursor):
- Queremos tratar solo los clientes que hayan invertido, en total, más de 50.000 €
- Para cada cliente que cumpla lo anterior:
  - Operación: Sumarle 100 € a la cantidad de cada una de sus inversiones (actualizando la tabla **Invierte**).
  - Si después de la operación, la suma de la cantidad de todas sus inversiones es superior a 1.000.000 €, deshacemos la actualización anterior y seguir con el siguiente cliente (usa savepoints).
  - Si después de la operación, la suma de la cantidad de todas sus inversiones es inferior a 100.000 €, entonces se creará una nueva compra en la tabla **Compras_XXXXX** de ese cliente con NumT = ‘999999’, NumFac = 0, Fecha la del sistema (sydate), importe = el 10 % de la suma obtenida anteriormente de cantidad en sus inversiones (en negativo), y Tienda = ‘bonus’.
  - Usa el nivel aislamiento adecuado para que la transacción actual no vea ninguna actualización de otras transacciones hasta que termine la transacción actual.
  - Haz lo necesario para que nadie pueda modificar la tabla **Compras_XXXXX** mientras estamos en el cliente **XXXXX** actual, pero que se libere dicha tabla al terminar (la transacción) con ese cliente y pasar al siguiente.

```
create or replace PROCEDURE EJ2  AS
  TDNI Invierte.dni%TYPE;
  Tsuma Number;
  Cursor CBonus is  select dni
                 from invierte
                 group by dni
                 having sum(Cantidad) > 50000;

BEGIN
 FOR cadaCli in CBonus
 LOOP
  SET transaction Isolation Level SERIALIZABLE;    /* read commited   tambiÃ©n*/
                          ---> comprueba permite poner aquÃ- SET??  */
  TDNI := cadaCli.dni;
  SAVEPOINT deshace;
  EXECUTE IMMEDIATE 'BEGIN
    LOCK TABLE  compras_' || TDNI || ' IN EXCLUSIVE MODE; END;';

  update invierte
    set cantidad = cantidad + 100
  where dni = TDNI;

  select sum(cantidad)  into Tsuma FROM invierte where DNI = TDNI;
  IF Tsuma > 1000000 THEN
      ROLLBACK to SAVEPOINT deshace;
```

```
      ELSEIF Tsuma < 100000 THEN
          EXECUTE IMMEDIATE 'BEGIN INSERT INTO compras_' || TDNI ||
               ' VALUES(''''particular'''',0, sysdate, ''''bonus'''', (Tsuma* -0,1) ); END;';
      END IF;
      commit; --- para liberar la tabla terminamos transacciÃ³n
    END LOOP;
    END;
```

b) Escribe un trigger para que en caso de cualquier actualización en la tabla invierte, se incluya una fila en una tabla *log_invierte* ya creada con la misma estructura que inverte, solo que con un atributo más llamado "operación", al que le asignará el valor correspondiente: *LOG*. Aunque la acualización en invierte se deshaga en el procedimiento EJ2, la fila en *log_invierte* debe conservarse, sin deshacerse.

```
create or replace TRIGGER trigger_EJ2
AFTER UPDATE ON invierte
FOR EACH ROW
  PRAGMA AUTONOMOUS TRANSACTION
BEGIN
      INSERT into log_invierte VALUES ('LOG',:old.DNI,  :old.NombreE,
                                                        :old.Cantidad, :old.Tipo);
commit;
END;
```

c) En la consulta `select DNI from cliente where NombreC <  Pepito';` ¿Usará el índice del DNI para hacer la consulta? Razona la respuesta.

NO, porque la condición no es sobre DNI, sino por NombreC

d) d.1) Qué hace la operación **`Index range scan`** si estuviera en el plan de ejecución de la consulta anterior? **d.2)** En general**,** cuándo se usa esa operación? **d.3)** Qué situación debe suceder para que en la consulta anterior esté esa operación?

D.1) **Accede a varias entradas de índice. D.2) Se aplica cuando hay condiciones de no igualdad o el índice no es único. D.3) Si el NombreC tuviera un índice.**

e) Hemos decidido hacer un índice en la tabla **`Invierte`**, sobre el atributo **`Tipo`** . Hay solo quince tipos diferentes. Qué clase de índice crearías? Escribe la instrucción para crearlo.

```
CREATE BITMAP INDEX idx_Tipo ON invierte (Tipo);
```

EJERCICIO 3 (1 punto)

a) Dadas estas tablas de una BD de políticos:

`Politico(DNI,cargo1,cargo2,NomPartido,votosObtenidos,siglasPartido,MatriculaCoche)`
`Partido(NomPartido,siglasPartido,NomComunidadAuto,siglasComunidadAuto,EscañosObtenidos)`

Se asume que cada candidato puede tener hasta dos cargos y un solo coche. Los partidos están en una o varias comunidades autónomas.

Se pide:

a.1) ¿Qué DFs problemáticas tiene cada tabla? Explica qué anomalías de actualización provocan.
a.2) ¿Qué PKs tiene cada tabla? Explica el porqué.
a.3) ¿En qué Forma Normal (la mejor) está cada tabla? Explica el porqué.
a.4) Que tipo de dependencia tienen **`votosObtenidos`** y **`EscañosObtenidos.`**

SOLUCION: **a.1) a.2) a.3)**

----- Politico **PK**: DNI
DFs problemáticas: NomPartido <--> siglasPartido : cada cambio, obliga a actualizar en
todas las filas los dos atributos de políticos de cada partido
➔ No cumple 3ª FN (transitiva) : está en 2ªFN

----- Partido **PK**: (NomPartido, NomComunidadAuto)

o bien (siglasPartido, siglasComunidadAuto)

-- DFs problemáticas

NomPartido --> siglasPartido : porque depende parcialmente de la PK

NomComunidadAuto --> siglasComunidadAuto : porque depende parcialmente de la PK

- cada cambio, obliga a actualizar en
todas las filas los dos atributos y vigilar que no haya discrepancias
entre los valores del nombre y las siglas

➔ No cumple 2ª FN (DF parcial de PK) : está en 1ªFN

a.4) Que tipo de dependencia tienen **votosObtenidos** y **EscañosObtenidos** :
NINGUNA dentro de las que hemos estudiado, porque están en dos tablas
Además votos son de candidato y escaños son de partido y autonomía