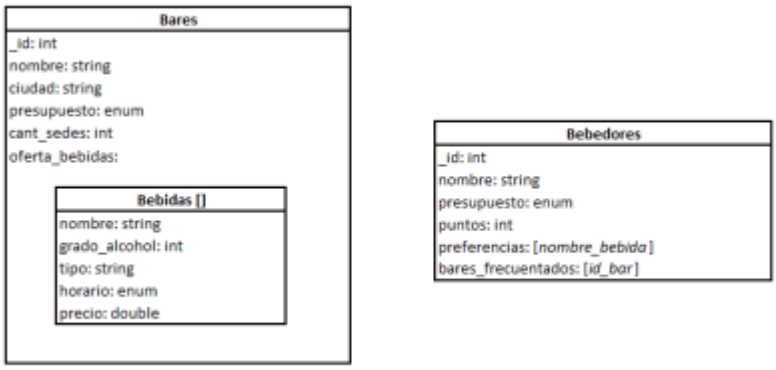
Taller 4 – Sistemas Transaccionales

Mauricio Martínez 202314461

Parte 1: Modelado de datos



1) ¿Qué estrategia se utilizó para modelar cada entidad de Parranderos dentro de la base de datos documental?
   - Bares se modeló con un objeto embebido llamado bebidas, que cuenta con un arreglo de todas las bebidas del bar. Y Bebedores se hizo de manera referencial, con las preferencias de bebida y bares que frecuenta el bebedor.

¿Cuáles relaciones se modelaron como objetos embebidos y cuáles se modelaron como referencias?

- Como objetos embebidos, se puede evidenciar la relación entre bares y bebidas, al Bebidas presentarse como un arreglo dentro de Bares, con toda la información de cada bebida con la que cuenta el bar. Y como referencia, se encuentra nombre_bebida, en las preferencias de los bebedores, y id_bar, que se encuentra como una referencia al id_bar, de cada uno de los bares frecuentados por el bebedor.

¿Qué relaciones y entidades del modelo se simplificaron?

- Se simplificó la necesidad de la tabla Sirven, Frecuentan y Gustan. Esto debido a que las relaciones entre dichas entidades, ya se encuentran tanto embebidas como referenciadas, en Bares y Bebedores, por lo que la necesidad de las tablas intermedias deja de existir.

2) ¿Cuáles son los tipos de consultas que podrían justificar las decisiones de modelado de Parranderos No relacional?
   - Serían consultas que disminuyen el tiempo, es decir, las que ya se encuentran embebidas dentro de la tabla.

¿Qué tipos de requerimientos de consulta se ven priorizados por el modelo?

- Las consultas que están directamente relacionadas con la entidad que se está consultando. Por ejemplo, el nombre de las bebidas que sirve un bar especifico.

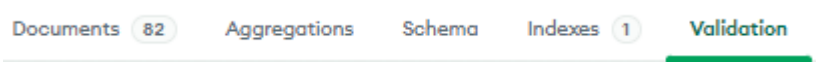
¿Qué tipo de requerimiento pierden prioridad? Mencione algunos ejemplos de requerimiento de consultas para justificar su respuesta.

- Los que impliquen hacer la relación de tablas para determinar la consulta. Un ejemplo, sería la de hallar el nombre de los bares que frecuenta un bebedor, esto debido a que no se encuentra embebido, por lo que toca unir tablas para realizar la consulta.

Parte 2: CRUD Básico

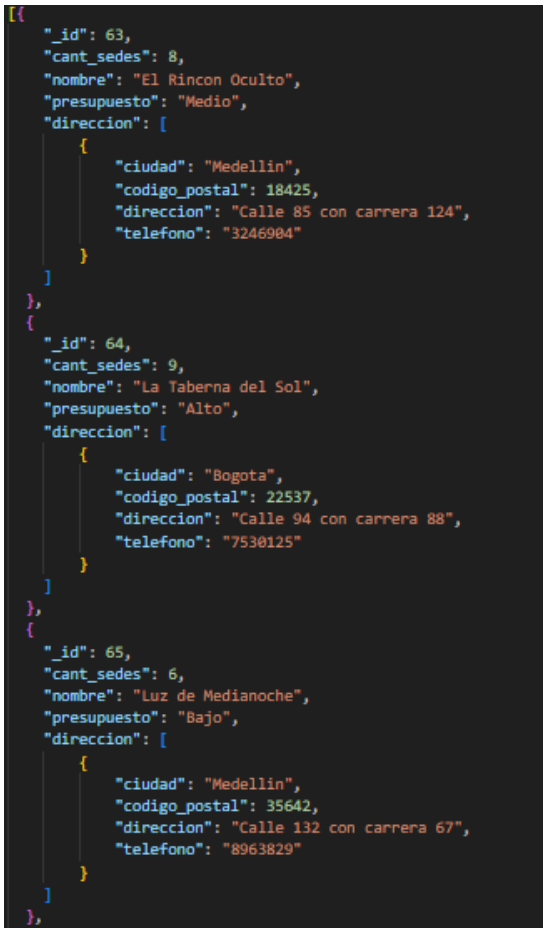
1) Realice las modificaciones necesarias al esquema existente de parranderos para reflejar este cambio. A la luz de la filosofía de modelado de datos para MongoDB, justifique su decisión.
   - Las modificaciones necesarias, serían deshacerse de objeto ciudad, en cada uno de los bares, para incluir este en un objeto embebido, que se conoce como dirección, en el cual, también se rellene con los datos de código_postal, dirección y teléfono.
2) Construya los esquemas de validación de los esquemas de la base de Parranderos que se ven afectados con este cambio.

   - El esquema de validación usado para Bares, con los nuevos cambios es el siguiente (Se ingresó el script del esquema de validación, en la pestaña "Validation"):



```
{
  $jsonSchema: {
    bsonType: 'object',
    required: [
      'nombre',
      'presupuesto',
      'cant_sedes',
      'direccion'
    ],
    properties: {
      nombre: {
        bsonType: 'string'
      },
      presupuesto: {
        bsonType: 'string'
      },
      cant_sedes: {
        bsonType: 'int'
      },
      direccion: {
        bsonType: 'object',
        required: [
          'ciudad',
          'codigo_postal',
          'direccion',
          'telefono'
        ],
        properties: {
          ciudad: {
            bsonType: 'string'
          },
          codigo_postal: {
            bsonType: 'int'
          },
          direccion: {
            bsonType: 'string'
          },
          telefono: {
            bsonType: 'string'
          }
        }
      }
    }
  }
}
```

3) Agregue a la base de datos al menos 20 nuevos bares que reflejen las modificaciones. Incluya en su informe el script de inserción utilizado.
   - Para agregar 20 nuevos bares que reflejen la modificación, se prefirió utilizar un archivo JSON, con la información de los 20 bares nuevos, una pequeña muestra de este archivo (junto a este informe, se encuentra adjunto el JSON):



```
[{
  "_id": 63,
  "cant_sedes": 8,
  "nombre": "El Rincon Oculto",
  "presupuesto": "Medio",
  "direccion": [
    {
      "ciudad": "Medellin",
      "codigo_postal": 18425,
      "direccion": "Calle 85 con carrera 124",
      "telefono": "3246904"
    }
  ]
 },
 {
  "_id": 64,
  "cant_sedes": 9,
  "nombre": "La Taberna del Sol",
  "presupuesto": "Alto",
  "direccion": [
    {
      "ciudad": "Bogota",
      "codigo_postal": 22537,
      "direccion": "Calle 94 con carrera 88",
      "telefono": "7530125"
    }
  ]
 },
 {
  "_id": 65,
  "cant_sedes": 6,
  "nombre": "Luz de Medianoche",
  "presupuesto": "Bajo",
  "direccion": [
    {
      "ciudad": "Medellin",
      "codigo_postal": 35642,
      "direccion": "Calle 132 con carrera 67",
      "telefono": "8963829"
    }
  ]
 },
```


4) Realice la actualización de todos los documentos de Mongo que no tengan dirección de acuerdo con la siguiente lista:
   • Ciudad: La ciudad original en donde está ubicada el bar.
   • Código postal: Un número aleatorio de 5 dígitos entre 10000 y 99999.
   • Direccion: Calle con carrera . Reemplace y por dos números aleatorios entre 1 y 150.
   • Teléfono: Un número de 7 dígitos aleatorio con el indicativo correspondiente a cada ciudad.

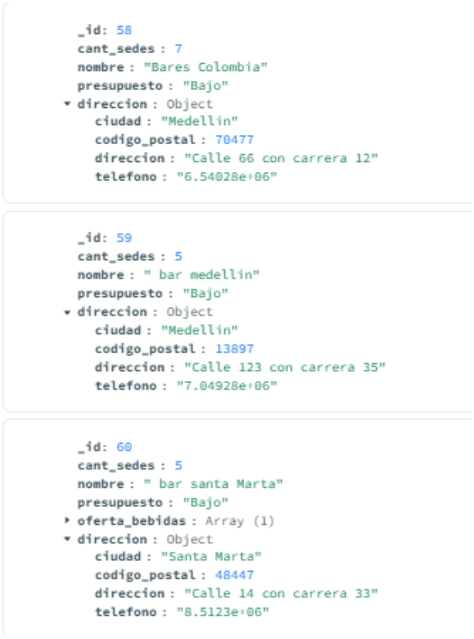
   - Para realizar está actualización, se verifica que la dirección no se encuentra en la tabla, para así no cambiar las direcciones ya existentes, y se hace un set para el objeto embebido dirección, en el cual ciudad es la ciudad existente, el código postal es un numero de 5 dígitos, Calle y carrera cumplen con el string especificado, y teléfono un numero de 7 digitos:

```
db["bares"].updateMany(
  { direccion: { $exists: false } },
  [
    {
      $set: {
        direccion: {
          ciudad: "$ciudad",
          codigo_postal: {$toInt: { $add: [10000, { $floor: { $multiply: [90000, { $rand: {} }] } }] }},
          direccion: {
            $concat: [
              "Calle ",
              { $toString: { $add: [1, { $floor: { $multiply: [150, { $rand: {} }] } }] } },
              " con carrera ",
              { $toString: { $add: [1, { $floor: { $multiply: [50, { $rand: {} }] } }] } }
            ]
          },
          telefono: { $toString: { $add: [1000000, { $floor: { $multiply: [9000000, { $rand: {} }] } }] } }
        }
      }
    }
  ]
);
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 62,
  modifiedCount: 62,
  upsertedCount: 0
}
```

- Después se eliminó ciudad, para las tablas que tienen este objeto:

```
db.bares.updateMany(
  {ciudad: { $exists: true }},
  { $unset: { ciudad: "" } }
);
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 62,
  modifiedCount: 62,
  upsertedCount: 0
}
```

- Este es el resultado, hubo un error con el número de teléfono, mostrando un valor que no es, pero lo demás se encuentra bien:

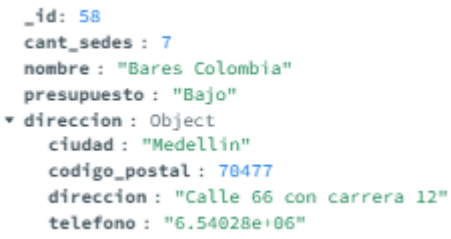


Parte 3: Consultas en bases de datos documentales

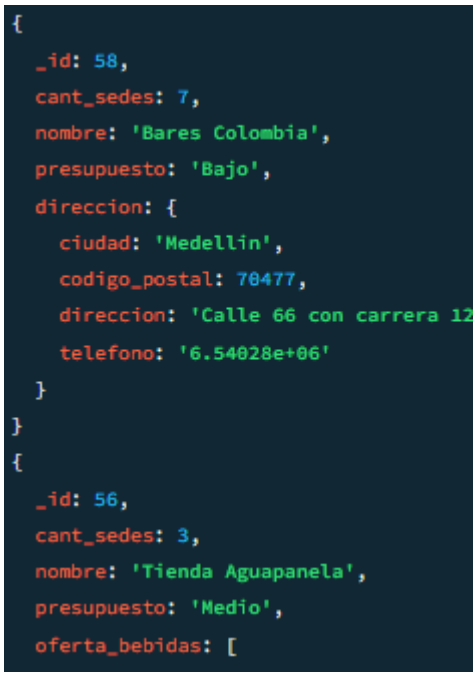
1) Se quieren conocer todos los bares que se encuentren en el mismo código postal que entre como parámetro
   - Se hace un find, a código_postal dentro del objeto embebido dirección, con el código postal solicitado:

```
db["bares"].find(
  { "direccion.codigo_postal": 70477 }
);
```

   - Usamos este de ejmplo:



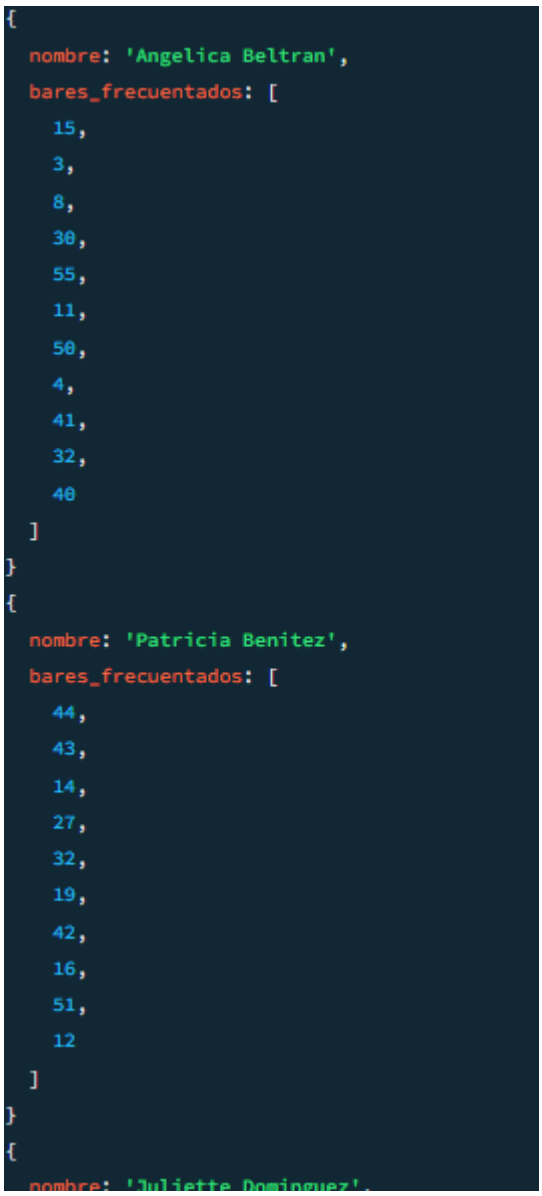
   - Se mostró tanto el _id 58, como el _id 56:



2) Se quieren conocer los nombres de bebedores que frecuentan entre 10 y 20 bares, junto con los ids de los bares que frecuentan.
   - Sobre la tabla bebedores se hace un match de los bares_frecuentados, con el tipo array, esto debido a que estaba dando error, ya que algunos bebedores no contaban con este objeto. Después se hace un project para que solo se tenga el nombre, los bares frecuentados, y cantidad de bares frecuentados (size de bares_frecuentados), después se hace un match para encontrar los que tienen total_bares, mayores o iguales a 10 (gte 10) y menores o iguales a 20 (lte 20). Y al final otro project, para solo mostrar el nombre del bebedor, y todos sus bares frecuentados (ids):

```
db["bebedores"].aggregate([
  {
    $match: {
      bares_frecuentados: { $type: "array" }
    }
  },
  {
    $project: {
      nombre: 1,
      bares_frecuentados: 1,
      total_bares: { $size: "$bares_frecuentados" }
    }
  },
  {
    $match: {
      total_bares: { $gte: 10, $lte: 20 }
    }
  },
  {
    $project: {
      nombre: 1,
      bares_frecuentados: 1,
      _id: 0
    }
  }
]);
{
```

- Este fue el resultado arrojado:



3) : Se quiere conocer para cada ciudad cuál es el tipo de bar más común que hay según su presupuesto (P.ej: esta consulta debería indicar que para Bogotá hay bares de presupuesto bajo).
   - Se agrupa primero por ciudad y presupuesto, y se halla el total de cada uno, después se ordena descendientemente, para hallar el más común, y se agrupa por la ciudad, y se halla el que está de primero (es decir, el de mayor cantidad), y se hace un project, con la ciudad y el presupuesto más común:

```
db["bares"].aggregate([
  {
    $group: {
      _id: { ciudad: "$direccion.ciudad", presupuesto: "$presupuesto" },
      total: { $sum: 1 }
    }
  },
  {
    $sort: { "_id.ciudad": 1, total: -1 }
  },
  {
    $group: {
      _id: "$_id.ciudad",
      presupuesto_mas_comun: { $first: "$_id.presupuesto" },
      total_bares: { $first: "$total" }
    }
  },
  {
    $project: {
      ciudad: "$_id",
      presupuesto_mas_comun: 1,
      _id: 0
    }
  }
]);
```