**Tarea 1:**

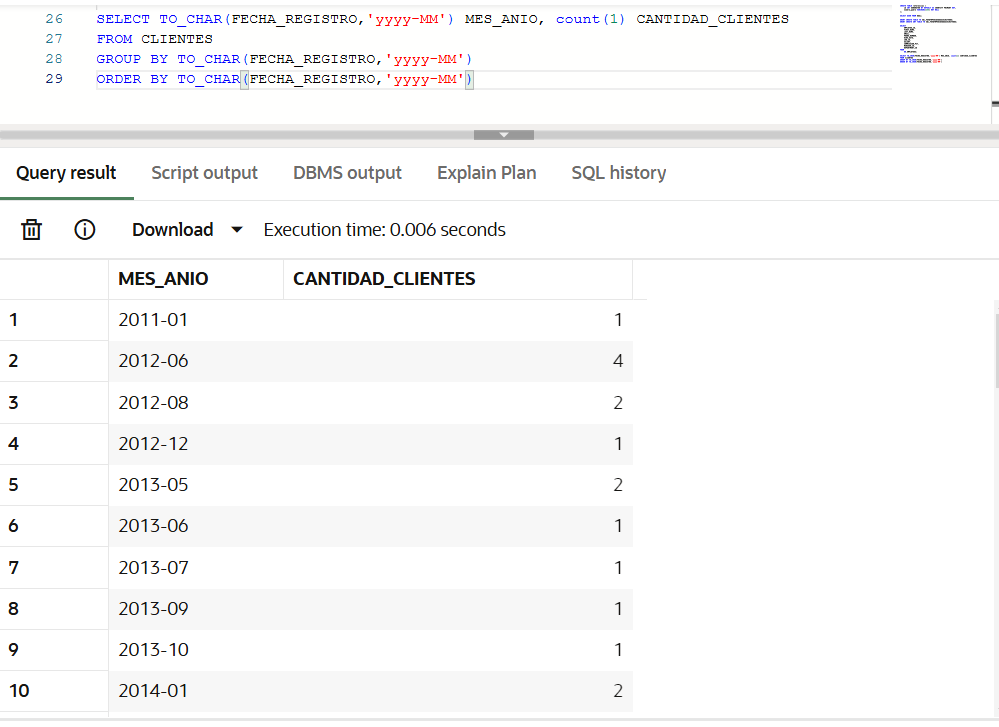
Tendencias de registro:

SELECT TO\_CHAR(FECHA\_REGISTRO,'yyyy-MM') MES\_ANIO, count(1) CANTIDAD\_CLIENTES

FROM CLIENTES

GROUP BY TO\_CHAR(FECHA\_REGISTRO,'yyyy-MM')

ORDER BY TO\_CHAR(FECHA\_REGISTRO,'yyyy-MM')



**Tarea 2:**

Jerarquía de clientes:

SELECT c1.ID, c1.NOMBRE, c2.NOMBRE AS NOMBRE\_PADRE,

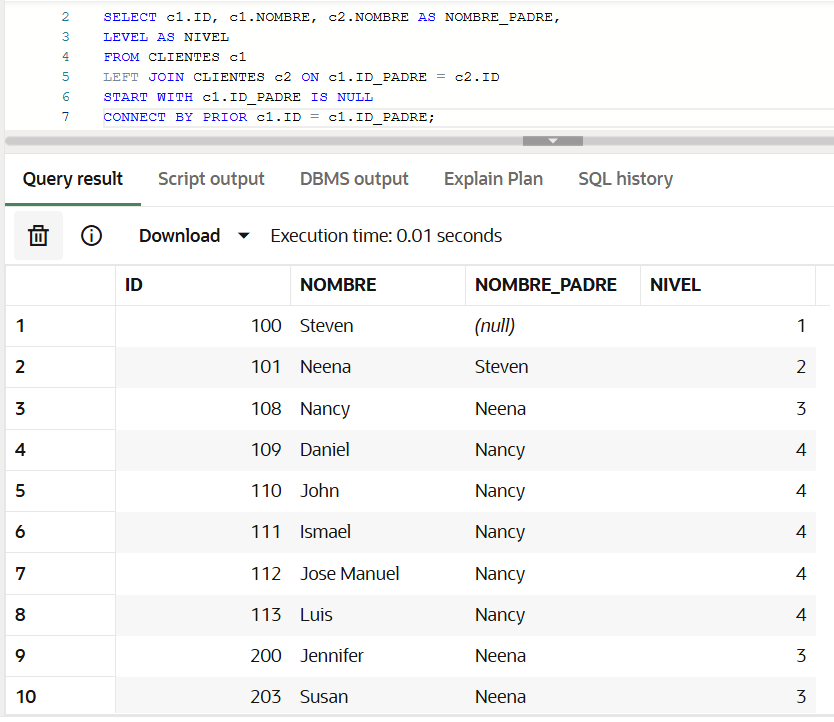
LEVEL AS NIVEL

FROM CLIENTES c1

LEFT JOIN CLIENTES c2 ON c1.ID\_PADRE = c2.ID

START WITH c1.ID\_PADRE IS NULL

CONNECT BY PRIOR c1.ID = c1.ID\_PADRE;

 **Tarea 3:**

Optimización de consultas

*Tendencia de registro*

El uso de TO\_CHAR en la primera consulta para poder obtener el formato YYYY-MM de una fecha de registro con formato YYYY-MM-DD tanto en el SELECT, ORDER BY, GROUP BY genera un posible cuello de botella dependiendo de la cantidad de registros a tener la tabla de clientes ya que se aplica la conversión a cada fila. Como se conoce, en cualquier base de datos el uso de ORDER BY afecta el rendimiento de la consulta.

Incluso al usar índices sobre la tabla puede afectar el rendimiento este tipo de conversiones.

En primera instancia el índice crear seria por FECHA\_REGISTRO, esto depende mucho de la forma en la que necesitemos realizar consultas sobre la tabla.

CREATE INDEX idx\_fecha\_registro ON CLIENTES (FECHA\_REGISTRO);

*Jerarquía de clientes*

Una de las formas de poder obtener el nombre del cliente (referido) se podía generar mediante un subquery en el SELECT y haciendo MATCH con la tabla principal, pero esto es ineficiente en caso de que la tabla de CLIENTES sea demasiado grande, así que la segunda opción fue realizar un LEFT JOIN con la misma tabla de CLIENTES para poder subir el valor de NOMBRE de la jerarquía.

Adicional a esto, se crea dos índices de los campos de ID y ID\_PADRE conforme al uso de los campos mas usados dentro de las consultas para mejorar las mismas.

CREATE INDEX idx\_cliente\_id ON CLIENTE (ID);

CREATE INDEX idx\_cliente\_id\_padre ON CLIENTE (ID\_PADRE);

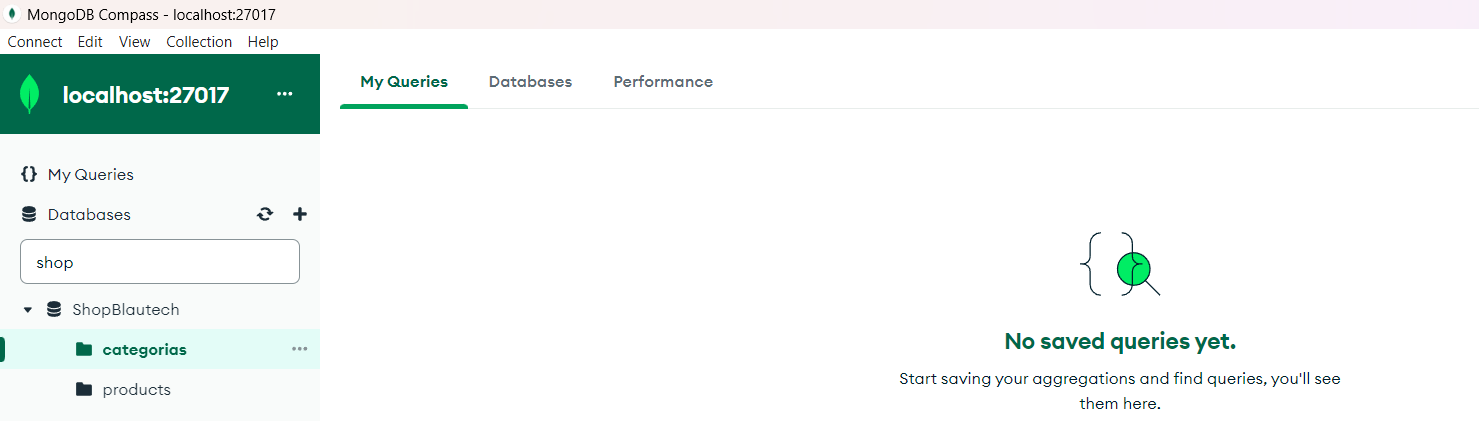
**Sección 2: Diseño y Optimización de Estructuras en MongoDB**

Diseño de esquemas

El diseño de esquema a utilizar para poder relacionar los productos con las categorías será de REFERENCING asumiendo (según el contexto bajo una previa validación) que estas relaciones no cambiaran con frecuencia.

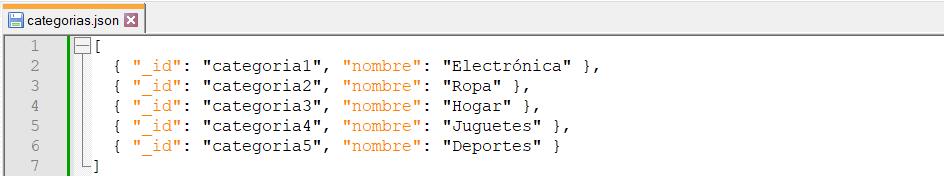
Se detalla el paso a paso para crear el esquema con sus respectivos documentos (de ejemplo)

* **Creación de collection**

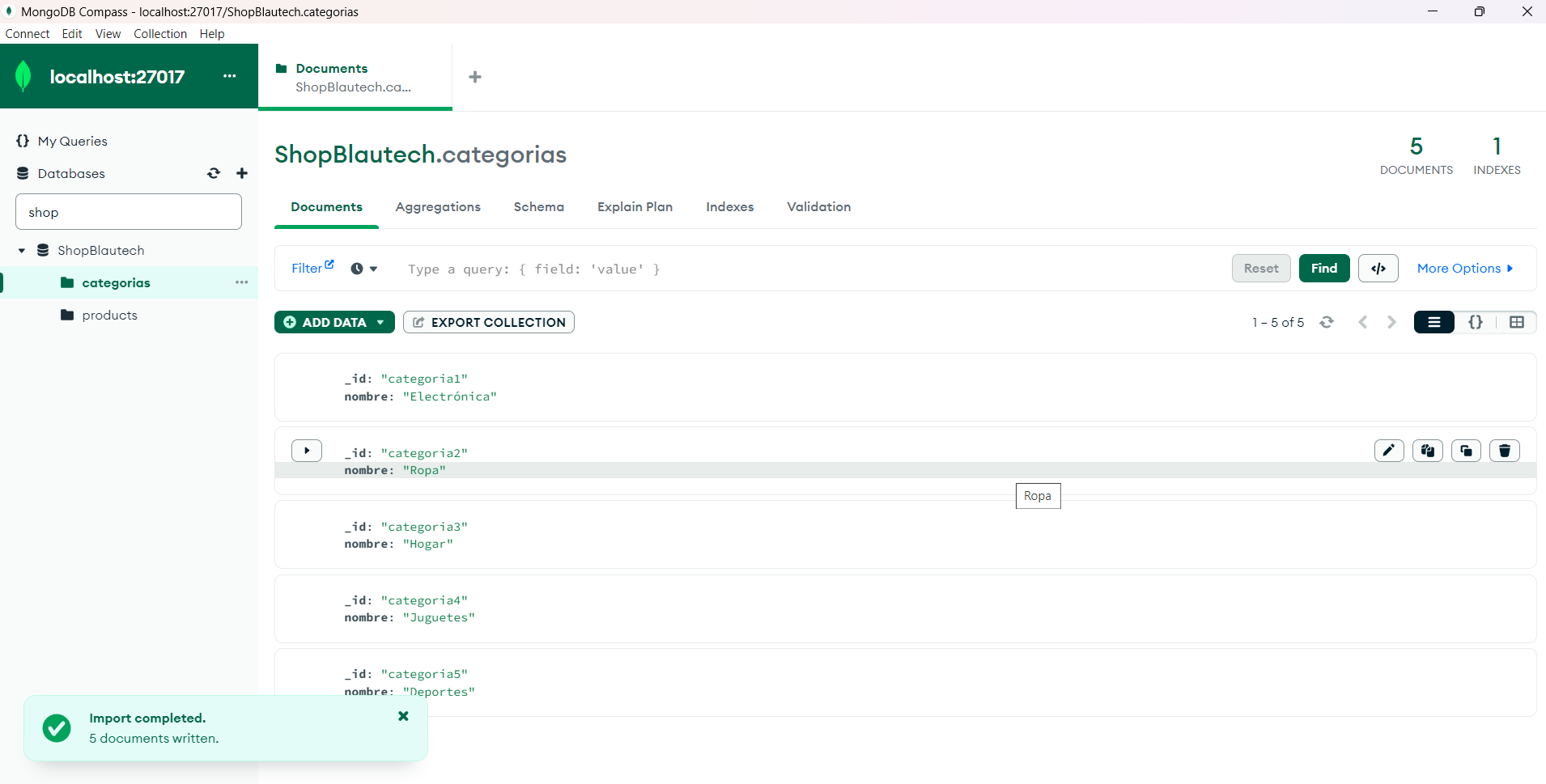
****

* **Insertar documentos a la colección de categorías**

Json creado con los documentos a importar

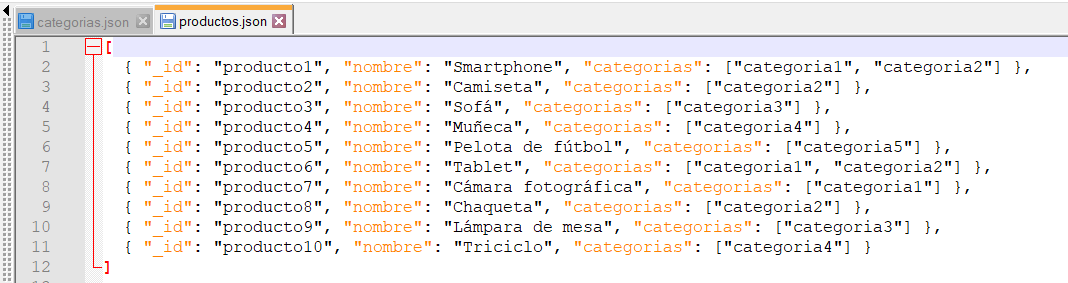


Con el apoyo del GUI de MongoDB importamos los datos del json

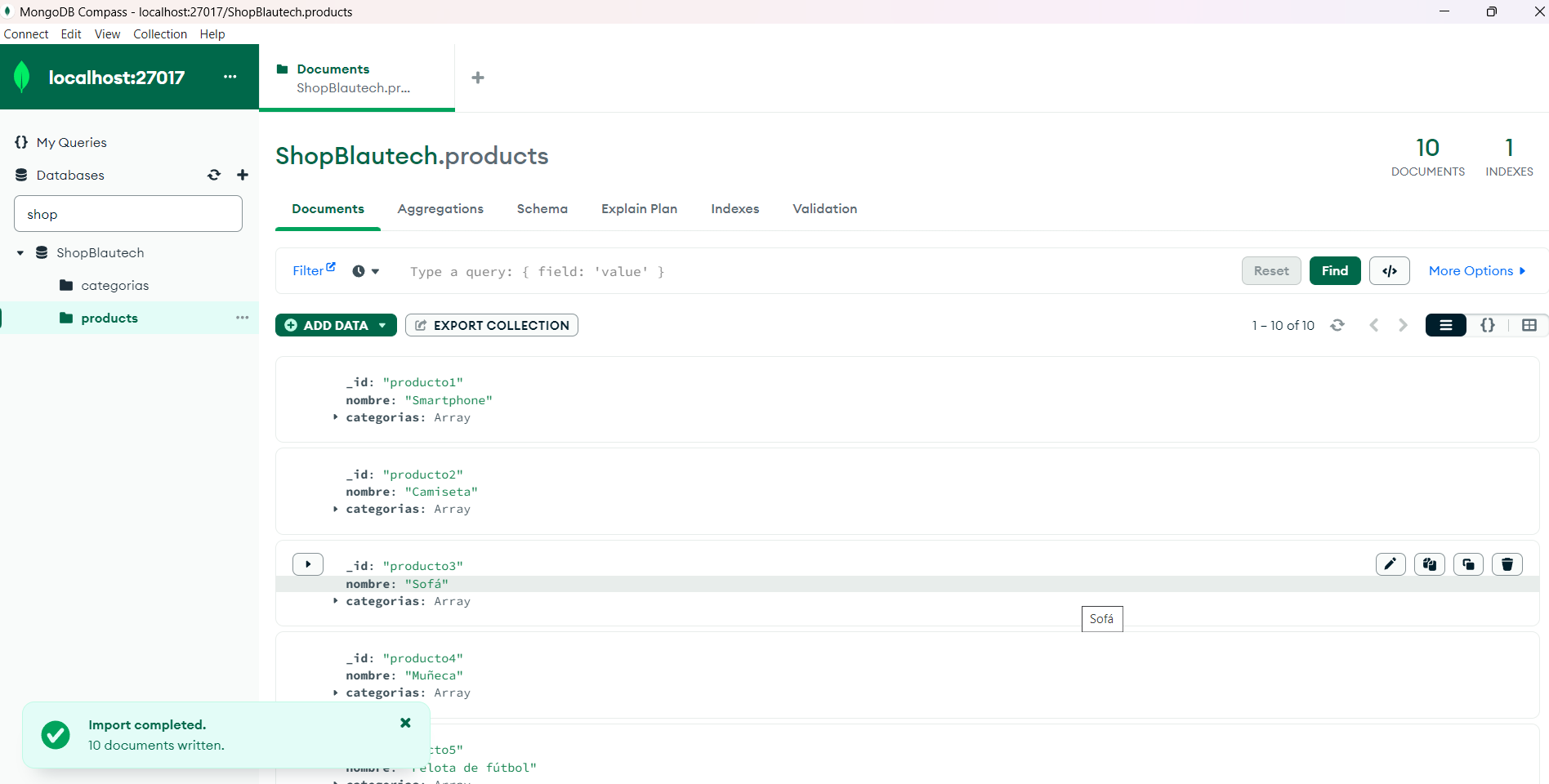


* **Insertar documentos a la colección de productos**

Json creado con los documentos a importar



Importación de documentos desde MongoDB



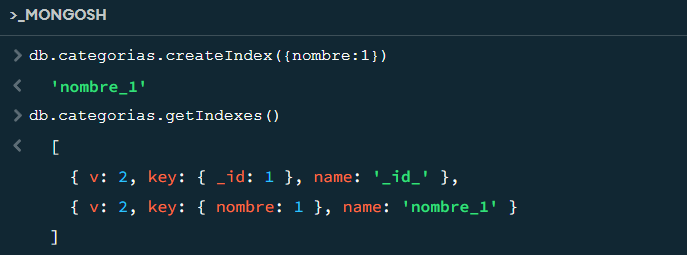
Optimización de consultas

Crear índices para las búsquedas más comunes como nombre de producto y nombre de categoría en orden ascendente. Para crear estos índices pueden ser desde la interfaz o mediante Shell.

db.productos.createIndex({ nombre: 1 })



db.categorias.createIndex({ nombre: 1 })



Uso de Explain Plan

*Productos*

**db.productos.find({ nombre: "Smartphone" }).explain("executionStats");**

Se marca en amarillo lo más relevante al EXPLAIN PLAN obtenido como: uso de índices, retorno de documentos, ejecución exitosa

{

  explainVersion: '1',

  queryPlanner: {

    namespace: 'ShopBlautech.productos',

    indexFilterSet: false,

    parsedQuery: {

      nombre: {

        '$eq': 'Smartphone'

      }

    },

    queryHash: '6DDCC174',

    planCacheKey: 'D8172F16',

    maxIndexedOrSolutionsReached: false,

    maxIndexedAndSolutionsReached: false,

    maxScansToExplodeReached: false,

    winningPlan: {

      stage: 'FETCH',

      inputStage: {

        stage: 'IXSCAN',

        keyPattern: {

          nombre: 1

        },

        indexName: 'nombre\_1',

        isMultiKey: false,

        multiKeyPaths: {

          nombre: []

        },

        isUnique: false,

        isSparse: false,

        isPartial: false,

        indexVersion: 2,

        direction: 'forward',

        indexBounds: {

          nombre: [

            '["Smartphone", "Smartphone"]'

          ]

        }

      }

    },

    rejectedPlans: []

  },

  executionStats: {

    executionSuccess: true,

    nReturned: 1,

    executionTimeMillis: 0,

    totalKeysExamined: 1,

    totalDocsExamined: 1,

    executionStages: {

      stage: 'FETCH',

      nReturned: 1,

      executionTimeMillisEstimate: 0,

      works: 2,

      advanced: 1,

      needTime: 0,

      needYield: 0,

      saveState: 0,

      restoreState: 0,

      isEOF: 1,

      docsExamined: 1,

      alreadyHasObj: 0,

      inputStage: {

        stage: 'IXSCAN',

        nReturned: 1,

        executionTimeMillisEstimate: 0,

        works: 2,

        advanced: 1,

        needTime: 0,

        needYield: 0,

        saveState: 0,

        restoreState: 0,

        isEOF: 1,

        keyPattern: {

          nombre: 1

        },

        indexName: 'nombre\_1',

        isMultiKey: false,

        multiKeyPaths: {

          nombre: []

        },

        isUnique: false,

        isSparse: false,

        isPartial: false,

        indexVersion: 2,

        direction: 'forward',

        indexBounds: {

          nombre: [

            '["Smartphone", "Smartphone"]'

          ]

        },

        keysExamined: 1,

        seeks: 1,

        dupsTested: 0,

        dupsDropped: 0

      }

    }

  },

  command: {

    find: 'productos',

    filter: {

      nombre: 'Smartphone'

    },

    '$db': 'ShopBlautech'

  },

  serverInfo: {

    host: 'LCALDERON-PC',

    port: 27017,

    version: '6.0.3',

    gitVersion: 'f803681c3ae19817d31958965850193de067c516'

  },

  serverParameters: {

    internalQueryFacetBufferSizeBytes: 104857600,

    internalQueryFacetMaxOutputDocSizeBytes: 104857600,

    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,

    internalDocumentSourceGroupMaxMemoryBytes: 104857600,

    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,

    internalQueryProhibitBlockingMergeOnMongoS: 0,

    internalQueryMaxAddToSetBytes: 104857600,

    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600

  },

  ok: 1

}

*Categorias*

**db.categorias.find({ nombre: "Deportes" }).explain("executionStats");**

Se marca en amarillo lo más relevante al EXPLAIN PLAN obtenido como: uso de índices, retorno de documentos, ejecución exitosa

{

  explainVersion: '1',

  queryPlanner: {

    namespace: 'ShopBlautech.categorias',

    indexFilterSet: false,

    parsedQuery: {

      nombre: {

        '$eq': 'Deportes'

      }

    },

    queryHash: '6DDCC174',

    planCacheKey: 'D8172F16',

    maxIndexedOrSolutionsReached: false,

    maxIndexedAndSolutionsReached: false,

    maxScansToExplodeReached: false,

    winningPlan: {

      stage: 'FETCH',

      inputStage: {

        stage: 'IXSCAN',

        keyPattern: {

          nombre: 1

        },

        indexName: 'nombre\_1',

        isMultiKey: false,

        multiKeyPaths: {

          nombre: []

        },

        isUnique: false,

        isSparse: false,

        isPartial: false,

        indexVersion: 2,

        direction: 'forward',

        indexBounds: {

          nombre: [

            '["Deportes", "Deportes"]'

          ]

        }

      }

    },

    rejectedPlans: []

  },

  executionStats: {

    executionSuccess: true,

    nReturned: 1,

    executionTimeMillis: 0,

    totalKeysExamined: 1,

    totalDocsExamined: 1,

    executionStages: {

      stage: 'FETCH',

      nReturned: 1,

      executionTimeMillisEstimate: 0,

      works: 2,

      advanced: 1,

      needTime: 0,

      needYield: 0,

      saveState: 0,

      restoreState: 0,

      isEOF: 1,

      docsExamined: 1,

      alreadyHasObj: 0,

      inputStage: {

        stage: 'IXSCAN',

        nReturned: 1,

        executionTimeMillisEstimate: 0,

        works: 2,

        advanced: 1,

        needTime: 0,

        needYield: 0,

        saveState: 0,

        restoreState: 0,

        isEOF: 1,

        keyPattern: {

          nombre: 1

        },

        indexName: 'nombre\_1',

        isMultiKey: false,

        multiKeyPaths: {

          nombre: []

        },

        isUnique: false,

        isSparse: false,

        isPartial: false,

        indexVersion: 2,

        direction: 'forward',

        indexBounds: {

          nombre: [

            '["Deportes", "Deportes"]'

          ]

        },

        keysExamined: 1,

        seeks: 1,

        dupsTested: 0,

        dupsDropped: 0

      }

    }

  },

  command: {

    find: 'categorias',

    filter: {

      nombre: 'Deportes'

    },

    '$db': 'ShopBlautech'

  },

  serverInfo: {

    host: 'LCALDERON-PC',

    port: 27017,

    version: '6.0.3',

    gitVersion: 'f803681c3ae19817d31958965850193de067c516'

  },

  serverParameters: {

    internalQueryFacetBufferSizeBytes: 104857600,

    internalQueryFacetMaxOutputDocSizeBytes: 104857600,

    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,

    internalDocumentSourceGroupMaxMemoryBytes: 104857600,

    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,

    internalQueryProhibitBlockingMergeOnMongoS: 0,

    internalQueryMaxAddToSetBytes: 104857600,

    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600

  },

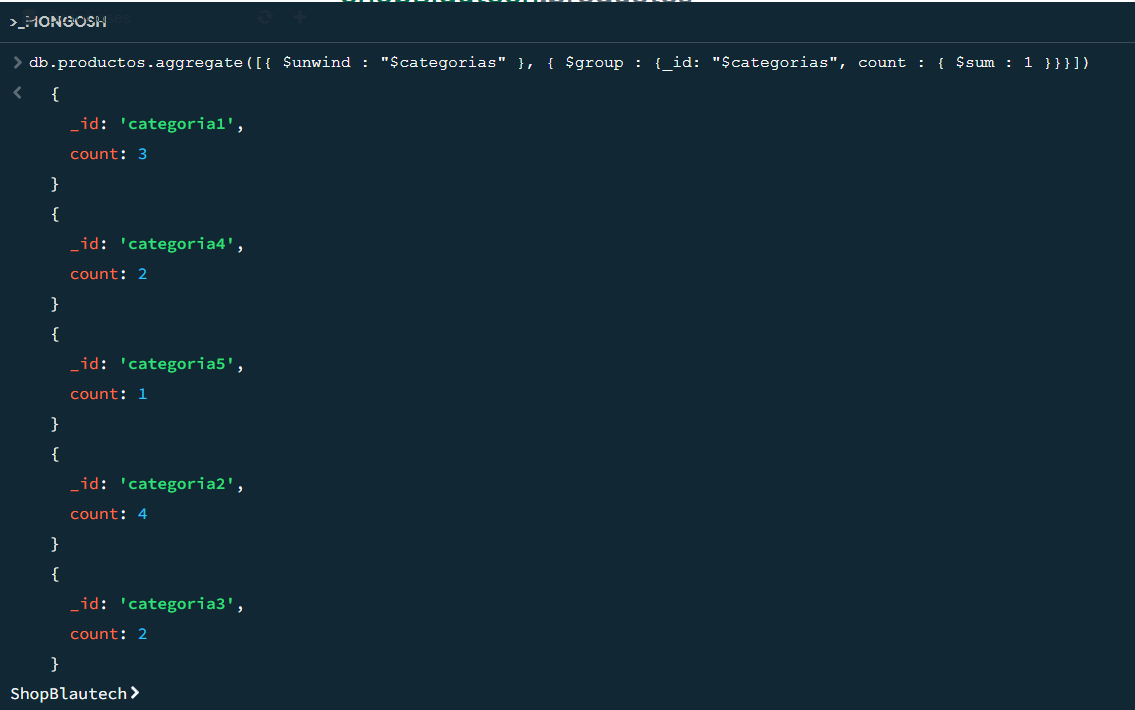
  ok: 1

}

Crear Agregaciones

Se requiere obtener el numero de productos asociados a cada categoria

db.productos.aggregate([{ $unwind : "$categorias" }, { $group : {\_id: "$categorias", count : { $sum : 1 }}}])



$unwind: Este operador toma un campo que es un arreglo (como una lista) y crea un nuevo documento para cada elemento de ese arreglo.

$group: Después de usar $unwind, se puede agrupar los documentos resultantes según un campo específico y realizar el conteo.