UNIVERSIDAD TECNOLÓGICA NACIONAL

Tecnicatura Universitaria en Programación

Informe Técnico: Operaciones CRUD en Colección Stations

Trabajo Práctico de Base de Datos NoSQL

Materia: Base de Datos II

Profesor: Ángel Simón

Alumno: Matías Ignacio Martorano

Legajo: 27526

Año 2025

1. Colección: stations

Almacena la información estática de las estaciones meteorológicas.

1.1. Estructura del Documento

```
{
  "_id": ObjectId("672f3a4d1b5e8a4a2c98d09f"),
  "Code": "89055",
  "OACI": "SAWB",
  "Name": "BASE MARAMBIO",
  "Province": "ANTARTIDA",
  "Latitude": -64.2333,
  "Longitude": -56.6167,
  "Altitude": 198
}
```

Listing 1: Ejemplo de documento completo en stations

2. Operaciones Implementadas

2.1. Insertar Documento Único (‘InsertOneAsync‘)

Utilizado para agregar una nueva estación manualmente.

- **C# (Driver):**

```
await Stations.InsertOneAsync(station);

```

- **Mongo Shell:**

```
db.stations.insertOne({
    "Code": "88963",
    "OACI": "SAYE",
    "Name": "BASE ESPERANZA",
    "Province": "ANTARTIDA",
    "Latitude": -63.3978,
    "Longitude": -56.9972,
    "Altitude": 24
})

```

2.2. Insertar Múltiples Documentos (‘InsertManyAsync‘)

Utilizado para la carga inicial masiva de estaciones.

- **C# (Driver):**

```
await Stations.InsertManyAsync(stations);

```

- **Mongo Shell:**

```
db.stations.insertMany([
    {
        "Code": "89034",
        "OACI": "SAYB",
        "Name": "BASE BELGRANO II",
        "Province": "ANTARTIDA",
        "Latitude": -77.8739,
        "Longitude": -34.6275,
        "Altitude": 256
    },
    {
        "Code": "89066",
        "OACI": "SAYS",
        "Name": "BASE SAN MARTIN",
        "Province": "ANTARTIDA",
        "Latitude": -68.1300,
        "Longitude": -67.1000,
        "Altitude": 7
    }
])

```

2.3. Leer Todos los Documentos (‘Find‘ sin filtros)

Utilizado para listar todas las estaciones disponibles.

- **C# (Driver):**

```
await Stations.Find(_ => true).ToListAsync();

```

- **Mongo Shell:**

```
db.stations.find({})

```

2.4. Leer un Documento Específico (‘Find‘ con filtro)

Utilizado para obtener los detalles de una estación por su ID.

- **C# (Driver):**

```
var filter = Builders<Station>.Filter.Eq(s => s.Id, id);
await Stations.Find(filter).FirstOrDefaultAsync();

```

- **Mongo Shell:**

```
db.stations.findOne({ "_id": ObjectId("672f3a4d1b5e8a4a2c98d09f") })

```

2.5. Actualizar un Documento (‘ReplaceOneAsync‘)

Reemplaza el documento completo con los nuevos valores editados.

- **C# (Driver):**

```
var filter = Builders<Station>.Filter.Eq(s => s.Id, station.Id);
await Stations.ReplaceOneAsync(filter, station);

```

- **Mongo Shell:**

```
db.stations.replaceOne(
    { "_id": ObjectId("672f3a4d1b5e8a4a2c98d09f") },
    {
        "Code": "89055",
        "OACI": "SAWB",
        "Name": "BASE MARAMBIO MODIFICADA",
        "Province": "ANTARTIDA",
        "Latitude": -64.2333,
        "Longitude": -56.6167,
        "Altitude": 200
    }
)

```

2.6. Eliminar un Documento (‘DeleteOneAsync‘)

Elimina una estación específica.

- **C# (Driver):**

```
var filter = Builders<Station>.Filter.Eq(s => s.Id, id);
await Stations.DeleteOneAsync(filter);

```

- **Mongo Shell:**

```
db.stations.deleteOne({ "_id": ObjectId("672f3a4d1b5e8a4a2c98d09f") })

```

2.7. Contar Documentos (‘CountDocumentsAsync‘)

Muestra el total de estaciones.

- **C# (Driver):**

```
await Stations.CountDocumentsAsync(_ => true);

```

- **Mongo Shell:**

```
db.stations.countDocuments({})

```

3. Colección: measurements

Esta colección almacena los datos meteorológicos horarios. Para optimizar el almacenamiento y permitir la normalización, cada documento de medición guarda una **referencia** a la estación meteorológica correspondiente, en lugar de duplicar toda la información de la estación.

3.1. Estructura del Documento

```
{
  "_id": ObjectId("672f3e8b1b5e8a4a2c98d0a1"),
  "station_id": ObjectId("672f3a4d1b5e8a4a2c98d09f"),
  "time": ISODate("2025-10-27T14:00:00.000Z"),
  "temp": -3.5,
  "dwpt": -4.9,
  "rhum": 90,
  "prcp": 0.0,
  "snow": null,
  "wdir": 25,
  "wspd": 14.8,
  "wpgt": null,
  "pres": 993.5,
  "tsun": null,
  "coco": 5
}
```

Listing 2: Ejemplo de documento en measurements

4. Operaciones Implementadas

4.1. Inserción Masiva de Datos (‘insertMany‘)

El colector de datos obtiene 24 mediciones de una sola vez por cada estación. Para eficiencia, estas se insertan en una sola operación por lotes.

- **C# (Driver):**

```
await Measurements.InsertManyAsync(measurements, new InsertManyOptions {
    IsOrdered = false });

```

- **Mongo Shell:**

```
db.measurements.insertMany([
   {
     "station_id": ObjectId("672f3a4d1b5e8a4a2c98d09f"),
     "time": ISODate("2025-11-08T10:00:00Z"),
     "temp": -5.4,
     "dwpt": -6.1,
     "rhum": 92,
     "prcp": 0.0,
     "snow": null,
     "wdir": 220,
     "wspd": 20.5,
     "wpgt": 35.0,
     "pres": 988.5,
     "tsun": null,
     "coco": 7
   },
   {
     "station_id": ObjectId("672f3a4d1b5e8a4a2c98d09f"),
     "time": ISODate("2025-11-08T11:00:00Z"),
```

```
    "temp": -5.1,
    "dwpt": -6.0,
    "rhum": 90,
    "prcp": 0.0,
    "snow": null,
    "wdir": 215,
    "wspd": 18.2,
    "wpgt": 32.0,
    "pres": 989.0,
    "tsun": null,
    "coco": 7
  }
], { ordered: false })

```

4.2. Inserción de Documento Único (‘insertOne‘)

Aunque menos común, el sistema permite insertar una medición individual si fuera necesario.

- **C# (Driver):**

```
await Measurements.InsertOneAsync(measurement);

```

- **Mongo Shell:**

```
db.measurements.insertOne({
    "station_id": ObjectId("672f3a4d1b5e8a4a2c98d09f"),
    "time": new Date(),
    "temp": -4.0,
    "dwpt": -5.5,
    "rhum": 85,
    "prcp": 0.0,
    "snow": null,
    "wdir": 210,
    "wspd": 15.0,
    "wpgt": 28.0,
    "pres": 990.1,
    "tsun": null,
    "coco": 3
})

```

4.3. Lectura de Todos los Documentos (‘find‘ sin filtros)

Utilizada para propósitos de depuración o volcado completo de datos.

- **C# (Driver):**

```
await Measurements.Find(_ => true).ToListAsync();

```

- **Mongo Shell:**

```
db.measurements.find({})

```

4.4. Consulta con Filtros Múltiples (‘find‘ / ‘*match*‘)

Utilizada por el módulo de gráficos para obtener los datos crudos antes de procesarlos. Permite filtrar por estación, rango de fechas y valores específicos de variables.

- **C# (Driver):**

```
var filter = Builders<Measurement>.Filter.Eq(m => m.StationId, ObjectId.
    Parse(stationId)) &
             Builders<Measurement>.Filter.Gte(m => m.Time, dateFrom) &
             Builders<Measurement>.Filter.Lte(m => m.Time, dateTo);

if (variableName != null && criteria != null && value.HasValue) {
    // Se agrega dinamicamente el filtro por variable
    // Ejemplo: filter &= builder.Gte("temp", -5.0);
}

await Measurements.Find(filter).ToListAsync();

```

- **Mongo Shell:**

```
db.measurements.find({
    "station_id": ObjectId("672f3a4d1b5e8a4a2c98d09f"),
    "time": {
        "$gte": ISODate("2025-11-01T00:00:00Z"),
        "$lte": ISODate("2025-11-08T23:59:59Z")
    },
    "temp": { "$gt": -5.0 }
})

```

4.5. Unión con Estaciones, Filtrado y Ordenamiento (Aggregation Pipeline)

Esta es la operación más compleja y completa del sistema, utilizada por la grilla principal de datos. Combina múltiples etapas del pipeline de agregación para obtener una vista enriquecida de los datos.

- **C# (Driver):**

```
var pipeline = Measurements.Aggregate()
    .Match(filter)          //$match (filtrado inicial)
    .Sort(sortDefinition)   //$sort (ordenamiento dinamico)
    .Lookup(                //$lookup (union con stations)
        foreignCollectionName: "stations",
        localField: "station_id",
        foreignField: "_id",
        as: "station_joined"
    )
    .Unwind("station_joined") // $unwind (aplanar array)
    .As<BsonDocument>();      // Proyeccion final

```

- **Mongo Shell:**

```
db.measurements.aggregate([
    {
        // Filtrar por estacion y fecha
        "$match": {
            "station_id": ObjectId("672f3a4d1b5e8a4a2c98d09f"),
            "time": {
                "$gte": ISODate("2025-11-01T00:00:00Z"),
                "$lte": ISODate("2025-11-08T23:59:59Z")
             }
        }
    },
    {
        // Ordenar por fecha descendente (mas reciente primero)
        "$sort": { "time": -1 }
    },
    {
        // Unir con la coleccion 'stations' para obtener el nombre
        "$lookup": {
            "from": "stations",
            "localField": "station_id",
            "foreignField": "_id",
            "as": "station_joined"
        }
    },
    {
        // 'Aplanar' el array resultante de la union para facilitar su uso
        "$unwind": "$station_joined"
    }
])

```

5. Colección: api_requests

Almacena un registro detallado de cada llamada realizada a la API externa de Meteostat. Esta colección es fundamental para la auditoría y el control de la cuota de uso mensual.

5.1. Estructura del Documento

```
{
  "_id": ObjectId("672f3b12a1b2c3d4e5f67890"),
  "station": {
    "_id": ObjectId("672f3a4d1b5e8a4a2c98d09f"),
    "Code": "89055",
    "OACI": "SAWB",
    "Name": "BASE MARAMBIO",
    "Province": "ANTARTIDA",
    "Latitude": -64.2333,
    "Longitude": -56.6167,
    "Altitude": 198
  },
  "timestamp": ISODate("2025-11-08T14:30:00Z"),
  "isScheduled": true,
  "success": true,
  "message": "OK"
}
```

Listing 3: Ejemplo de documento en api_requests

6. Operaciones Implementadas

6.1. Insertar Registro (‘InsertOneAsync‘)

Utilizado por el colector automático para registrar cada intento de conexión con la API, independientemente de si tuvo éxito o falló.

- **C# (Driver):**

```
await Logs.InsertOneAsync(log);

```

- **Mongo Shell:**

```
db.api_requests.insertOne({
    "station": {
        "_id": ObjectId("672f3a4d1b5e8a4a2c98d09f"),
        "Code": "89055",
        "Name": "BASE MARAMBIO",
        // ... Resto de los campos ...
    },
    "timestamp": new Date(),
    "isScheduled": true,
    "success": true,
    "message": "OK"
})

```

6.2. Leer Registros por Mes (‘Find‘ con filtro de fecha)

Utilizado para consultar el historial de llamadas realizadas en un período específico (por ejemplo, un mes calendario).

- **C# (Driver):**

```
var start = new DateTime(year, month, 1);
var end = start.AddMonths(1);
var filter = Builders<ApiRequestLog>.Filter.Gte(l => l.Timestamp, start) &
             Builders<ApiRequestLog>.Filter.Lt(l => l.Timestamp, end);

await Logs.Find(filter).ToListAsync();

```

- **Mongo Shell:**

```
db.api_requests.find({
    "timestamp": {
        "$gte": ISODate("2025-11-01T00:00:00Z"),
        "$lt": ISODate("2025-12-01T00:00:00Z")
    }
})

```

6.3. Contar Llamadas Mensuales (‘CountDocumentsAsync‘)

Operación crítica utilizada antes de cada ciclo de sondeo para verificar si se ha alcanzado el límite mensual de 500 llamadas a la API gratuita.

- **C# (Driver):**

```
var start = new DateTime(DateTime.Now.Year, DateTime.Now.Month, 1);
var end = start.AddMonths(1);
var filter = Builders<ApiRequestLog>.Filter.Gte(l => l.Timestamp, start) &
             Builders<ApiRequestLog>.Filter.Lt(l => l.Timestamp, end);

await Logs.CountDocumentsAsync(filter);

```

- **Mongo Shell:**

```
db.api_requests.countDocuments({
    "timestamp": {
        "$gte": ISODate("2025-11-01T00:00:00Z"),
        "$lt": ISODate("2025-12-01T00:00:00Z")
    }
})

```