**Resumen Ejecutivo**

Este informe documenta el diseño y desarrollo de una base de datos para un sistema de reservas de vuelos utilizando MongoDB. La base de datos incluye colecciones que permiten la gestión de usuarios, aeropuertos, vuelos y reservas, integrando relaciones uno a uno, uno a muchos, y muchos a muchos. El informe analiza la estructura de los datos, los métodos de captura utilizados y los resultados obtenidos a través de consultas y actualizaciones en MongoDB. Se describen las decisiones de diseño tomadas en cuanto a la normalización y la elección de embebido o referencia de documentos, lo que permite escalabilidad y eficiencia en las consultas.

**Introducción**

**Contexto y Motivación**

En la actualidad, los sistemas de gestión de reservas de vuelos requieren soluciones robustas y escalables que puedan manejar grandes volúmenes de datos de manera eficiente. MongoDB es una base de datos NoSQL que ofrece flexibilidad en el diseño y una estructura de documentos adecuada para manejar estos escenarios. El presente informe aborda la creación de una base de datos de reservas de vuelos en MongoDB, explicando las relaciones y estructuras empleadas en su diseño.

**Alcance del Informe**

Este informe cubre los siguientes aspectos de MongoDB:

* Diseño de la base de datos: estructura de colecciones y relaciones.
* Consultas de actualización de datos: uso de updateOne(), upsert, setOnInsert y otros métodos.
* Análisis de esquemas y normalización.
* Discusión sobre el diseño de las relaciones entre los datos.

**Objetivos**

Esta actividad te permitirá profundizar en el uso de MongoDB. Realizarás una serie de ejercicios guiados que te ayudarán a comprender mejor las operaciones del CRUD de esta base de datos documental, así como sus particularidades.

A través del desarrollo de la base de datos de reservas de vuelos, se busca:

1. Explorar las capacidades de MongoDB para la creación, lectura, actualización y eliminación de datos (CRUD).
2. Comprender las operaciones clave del CRUD en MongoDB:
   * **CREATE**: Añadir documentos a las colecciones con insertOne() y insertMany().
   * **READ**: Recuperar documentos con find() y findOne().
   * **UPDATE**: Modificar documentos existentes con updateOne() y updateMany().
   * **DELETE**: Eliminar documentos de las colecciones con deleteOne() y deleteMany().
3. Utilizar herramientas como MongoDB Compass o la consola de Mongo (mongosh) para realizar las operaciones necesarias.
4. Visualizar todas las bases de datos existentes usando el comando show dbs para comprender mejor el entorno de trabajo.

**Metodología**

**Herramientas Utilizadas**

* **MongoDB**: Base de datos NoSQL utilizada para almacenar y gestionar la información del sistema de reservas de vuelos.
* **Studio 3T**: Herramienta visual para la administración de MongoDB que facilita la consulta, manipulación y análisis de datos.
* **MongoDB Compass**: Herramienta gráfica para la gestión de bases de datos MongoDB.
* **JavaScript (shell MongoDB)**: Para la creación de consultas y scripts de actualización de datos.

**Procedimientos**

**1. Métodos y pasos seguidos para llevar a cabo el análisis o el trabajo**

* 1. **Creación base de datos personal**

La base de datos fue diseñada para gestionar la información clave relacionada con reservas de vuelos, incluyendo las siguientes colecciones: usuarios, aeropuertos, vuelos y reservas. Cada una de estas colecciones tiene un esquema claro que permite la relación entre ellas mediante IDs.

* 1. **Realización del taller**

Durante el desarrollo del taller se realizaron las siguientes tareas:

* Definición de colecciones y sus relaciones.
* Implementación de métodos de actualización avanzados (updateOne(), upsert, setOnInsert).
* Creación de datos de prueba para simular escenarios reales de uso.

**1.2.1 Métodos de captura**

Se utilizó Studio 3T y MongoDB Compass para visualizar los esquemas y relaciones, además de realizar consultas de prueba para validar la estructura y coherencia de los datos. Las consultas se realizaron directamente desde la interfaz de Studio 3T y desde la shell de MongoDB para evaluar el rendimiento.

**Desarrollo del Informe**

**Descripción de la Base de Datos Personal**

La base de datos diseñada contiene cuatro colecciones principales: usuarios, aeropuertos, vuelos y reservas. A continuación se muestra una breve descripción de cada colección:

1. **Usuarios**: Almacena la información personal de los clientes, como nombre, correo, pasaporte, teléfono y dirección.
2. **Aeropuertos**: Almacena los detalles de los aeropuertos, incluyendo el código IATA, nombre, ciudad, país, capacidad anual y número de terminales.
3. **Vuelos**: Contiene los datos relacionados con los vuelos, como origen, destino, fecha, duración, asientos disponibles, precio, aerolínea y número de vuelo.
4. **Reservas**: Relaciona a los usuarios con los vuelos, almacenando la información de la reserva, incluyendo asientos reservados, estado y método de pago.

**Diseño de Base de Datos Personal**

1. **Modelo de Datos**

El diseño del modelo de datos sigue los principios de normalización en cuanto a la separación de entidades (colecciones) y el uso de referencias entre ellas. Por ejemplo, los usuarios y vuelos se mantienen en colecciones separadas y se relacionan a través de la colección de reservas, que almacena los IDs de los usuarios y los vuelos.

**Normalización**: Las entidades principales (usuarios, aeropuertos, vuelos) se mantienen en colecciones separadas, mientras que las relaciones (reservas) se manejan con referencias para evitar redundancias.

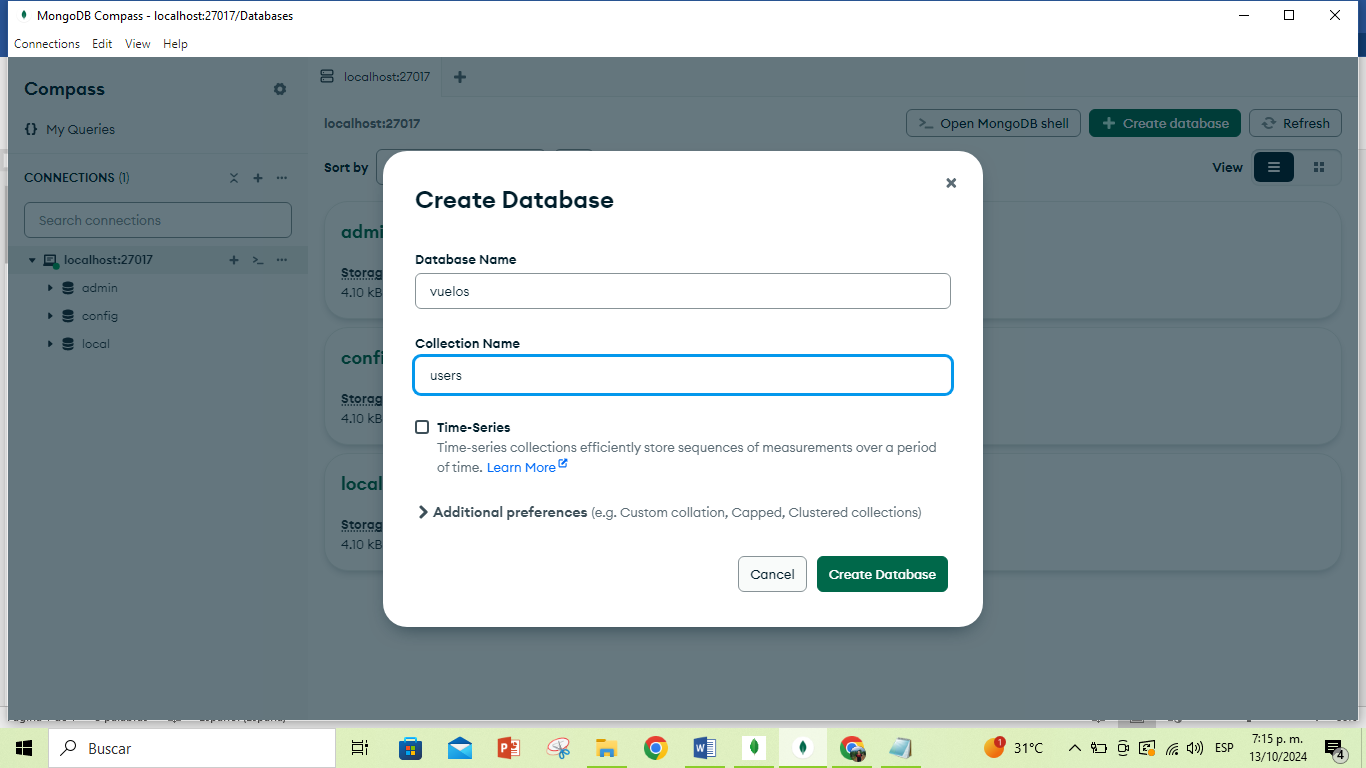
**Cardinalidad**:

* **Relaciones Uno a Uno (1:1)**: Un usuario tiene un pasaporte único.
* **Relaciones Uno a Muchos (1-M)**: Un vuelo puede tener múltiples reservas.
* **Relaciones Muchos a Muchos (M-M)**: Un usuario puede reservar múltiples vuelos, y un vuelo puede tener múltiples usuarios reservados.

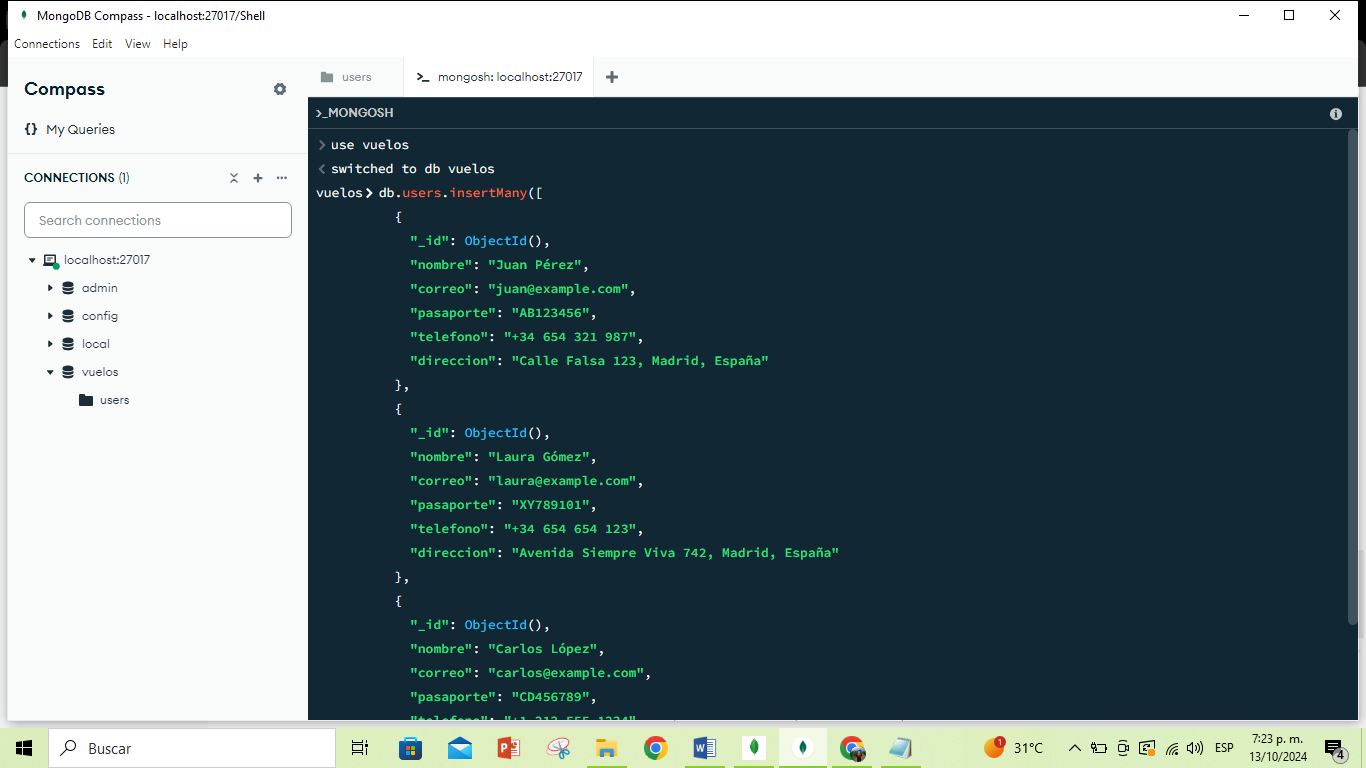
1. **Consideraciones de Diseño**

* **Relaciones entre colecciones**: Las relaciones se establecen utilizando referencias a través de los IDs de los documentos. Esto asegura que los datos se mantengan separados y que puedan ser accedidos de manera eficiente mediante consultas.
* **Definición de campos comunes**: Los campos como usuario\_id y vuelo\_id permiten conectar usuarios y vuelos a través de las reservas, asegurando una relación clara y concisa entre las colecciones.

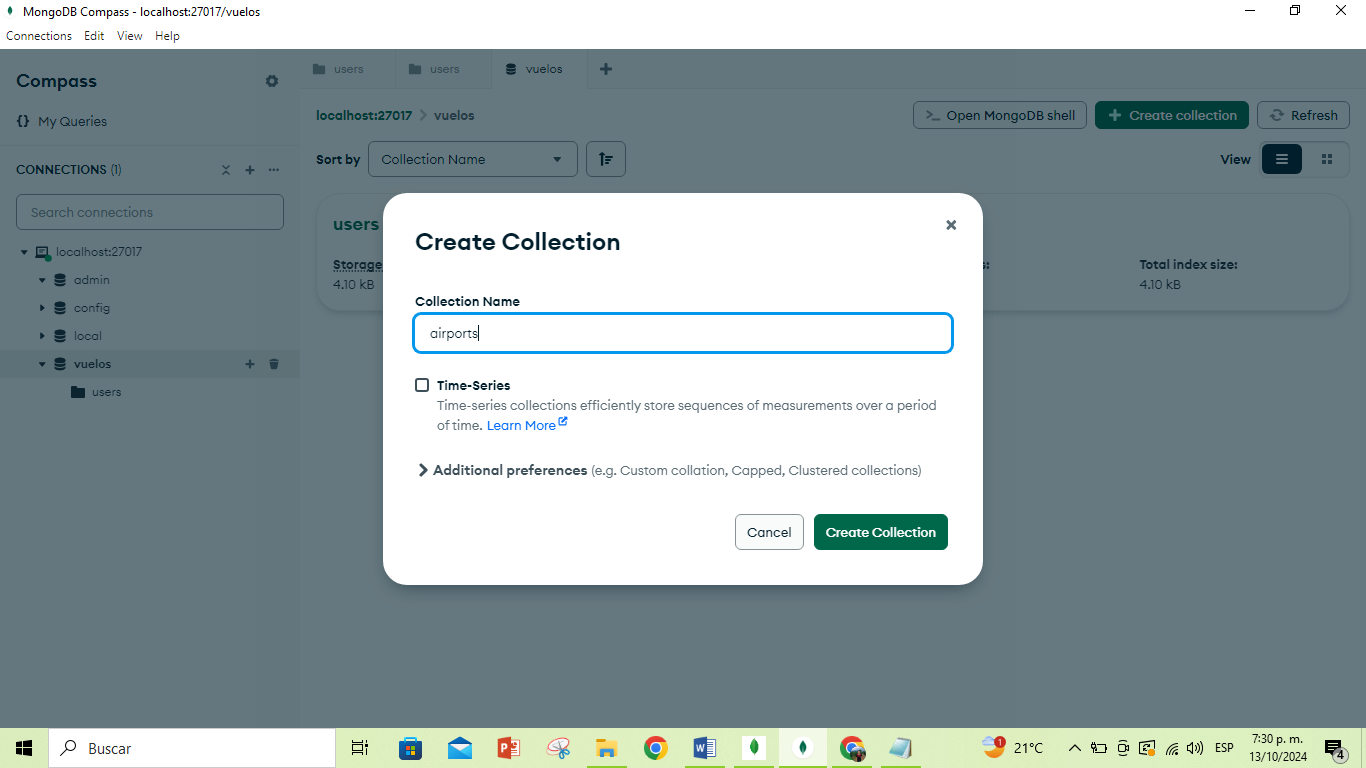
**Creación de base de datos**



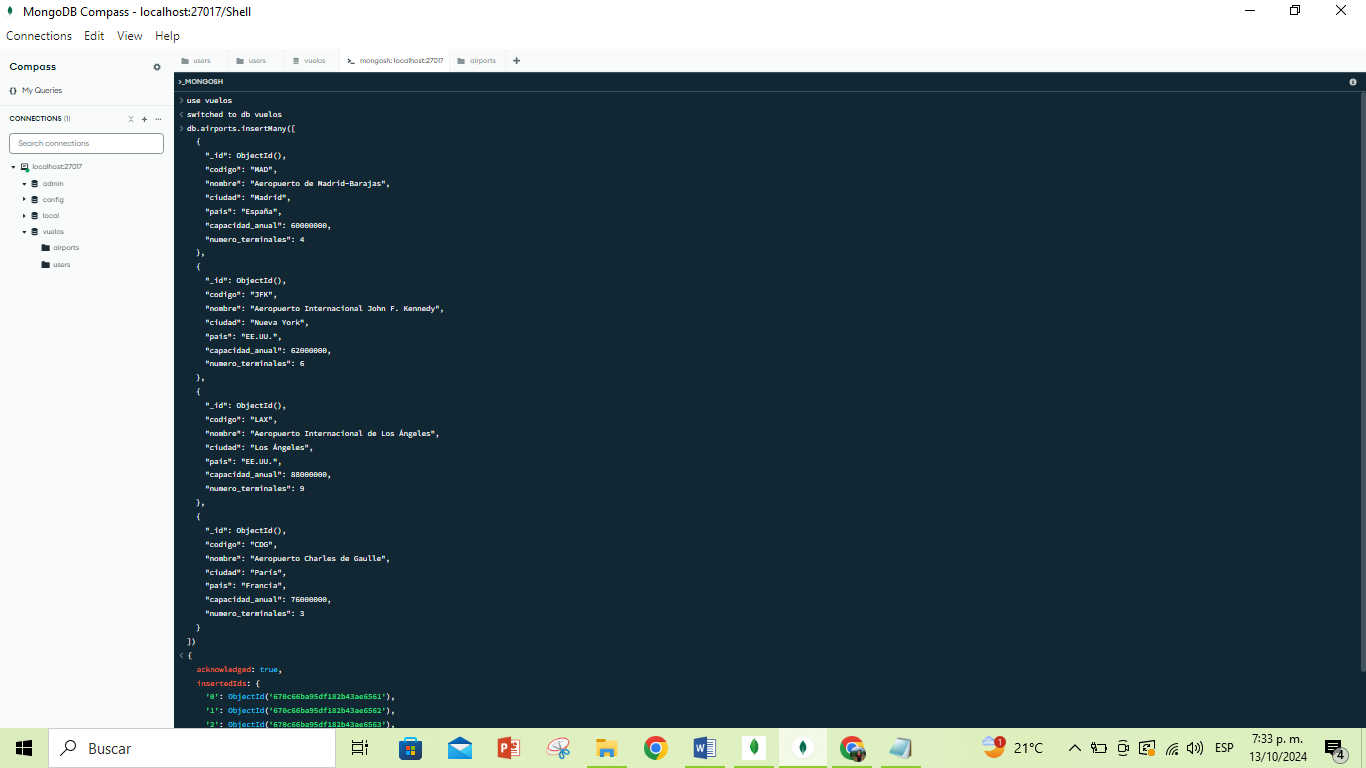
**Ceración de la collection users y sus datos**



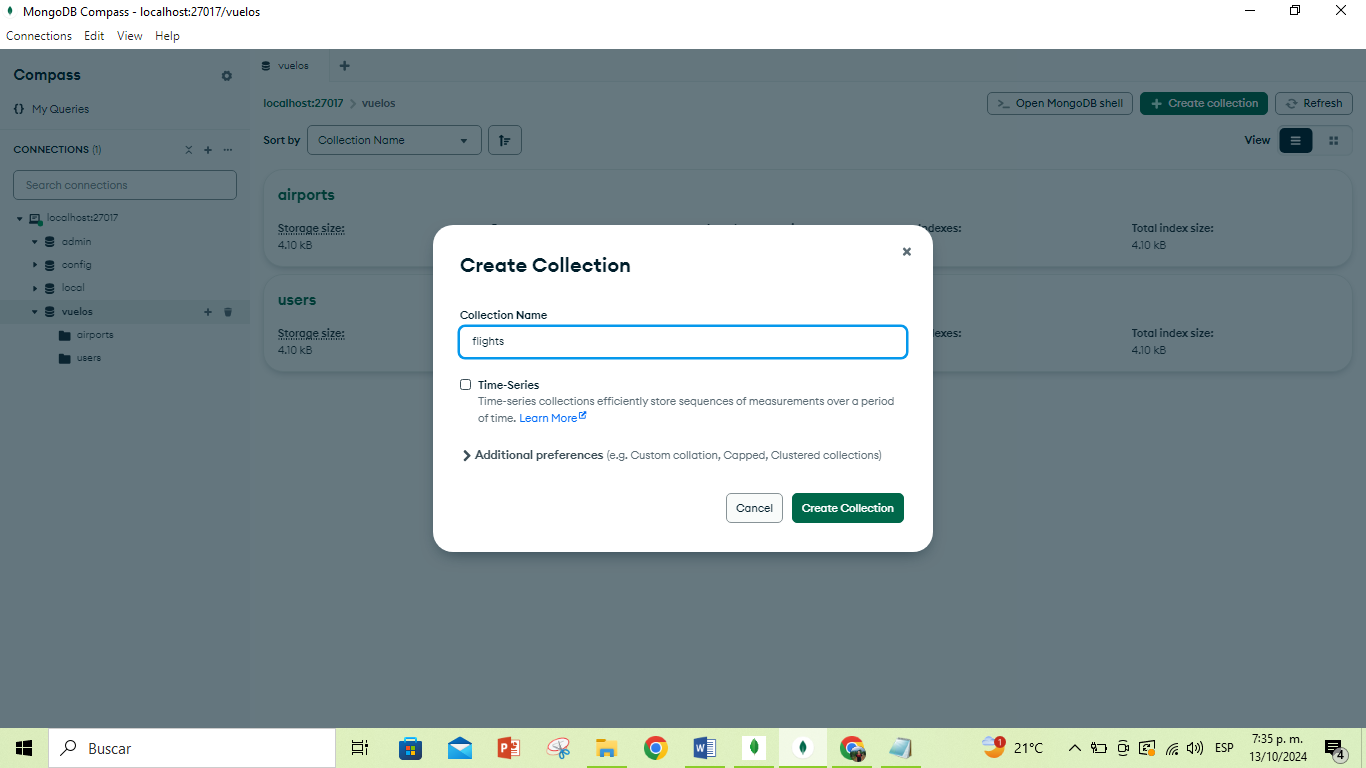
**Creación de la colecction airports**



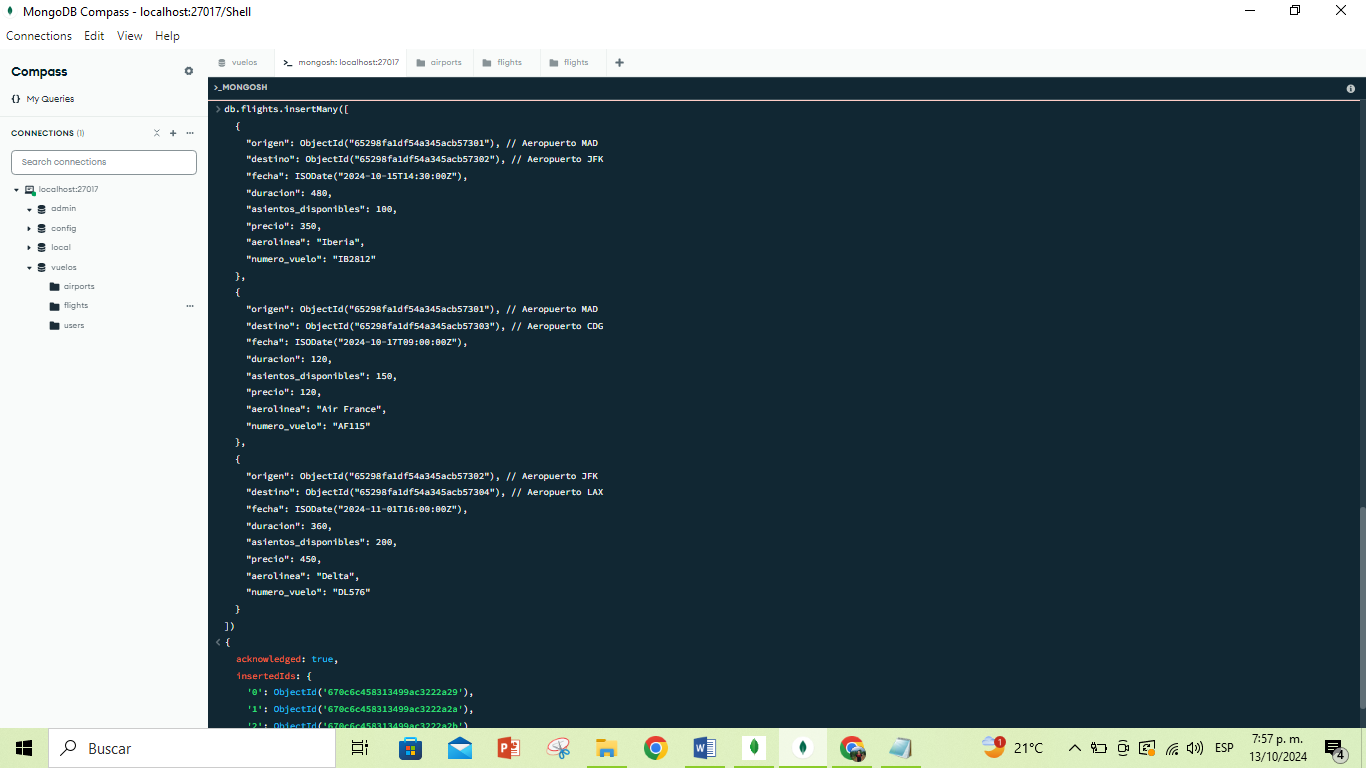
**Creación y llenado de datos de la collection airports**



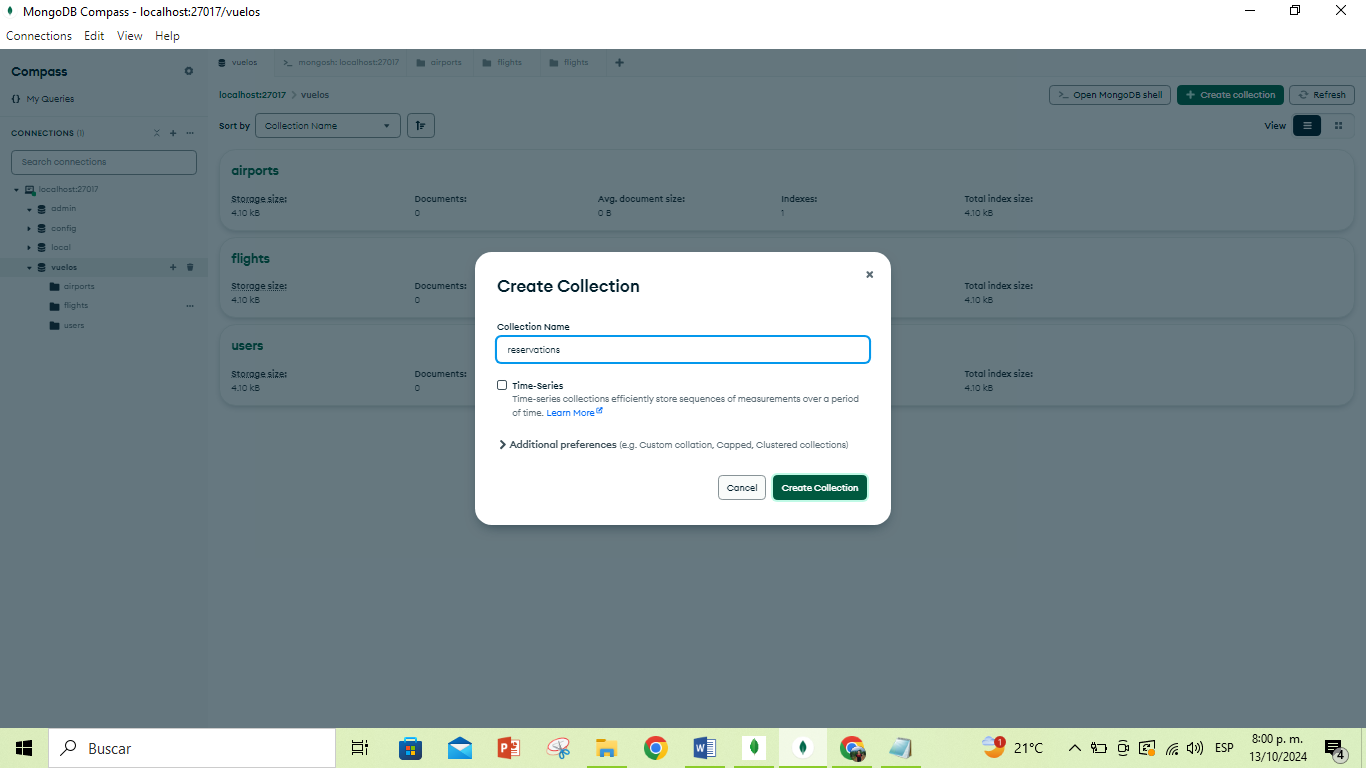
**Creación de la collection fligths**



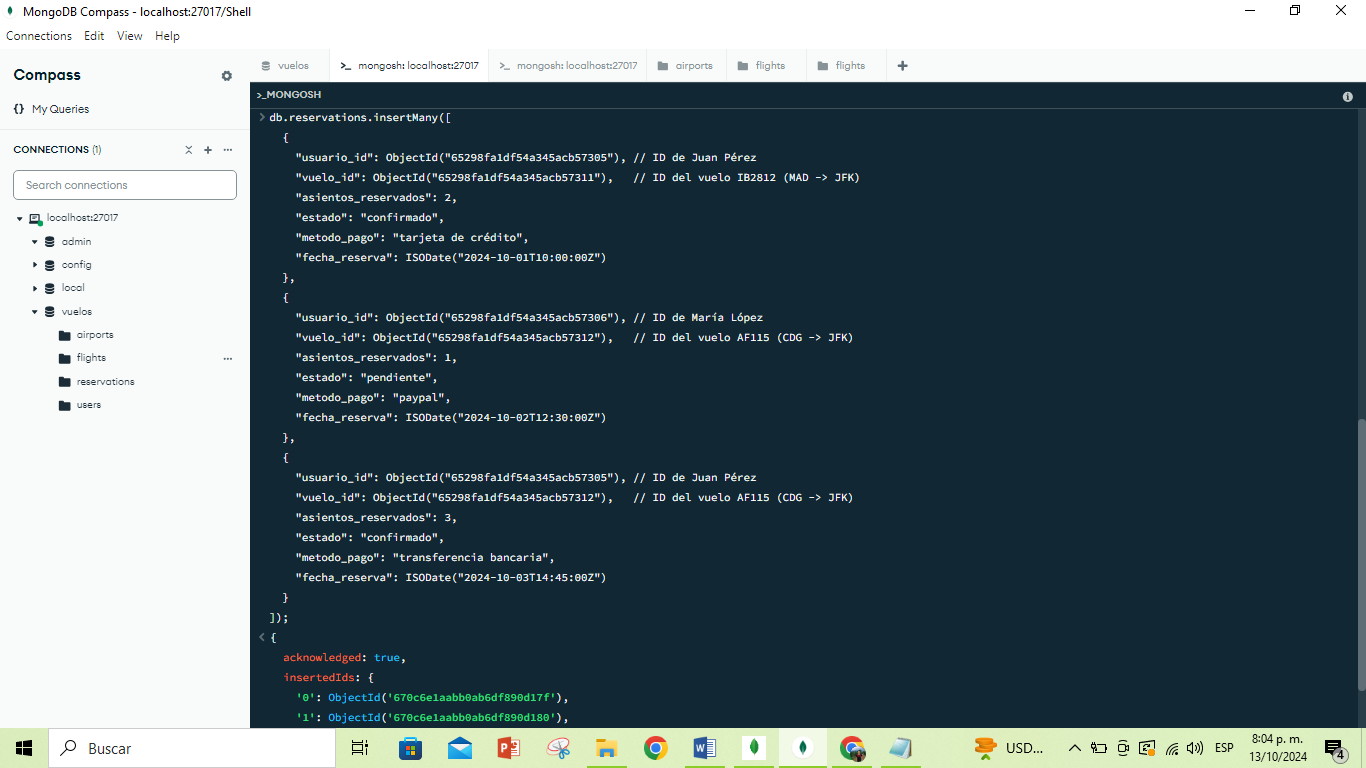
**Creación y llenado de datos de la colección flights**



**Creación de la collection reservations**



**Creación y llenado de datos de la colección reservations**

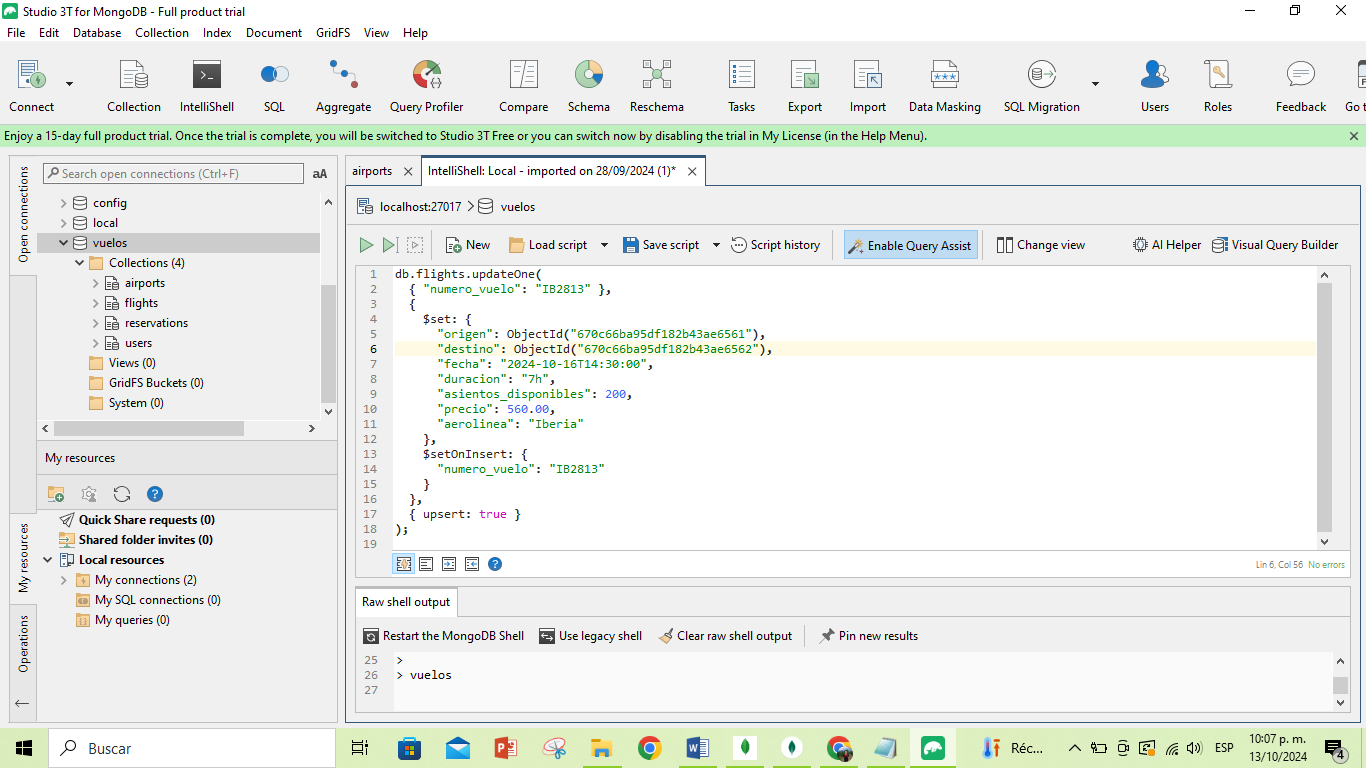


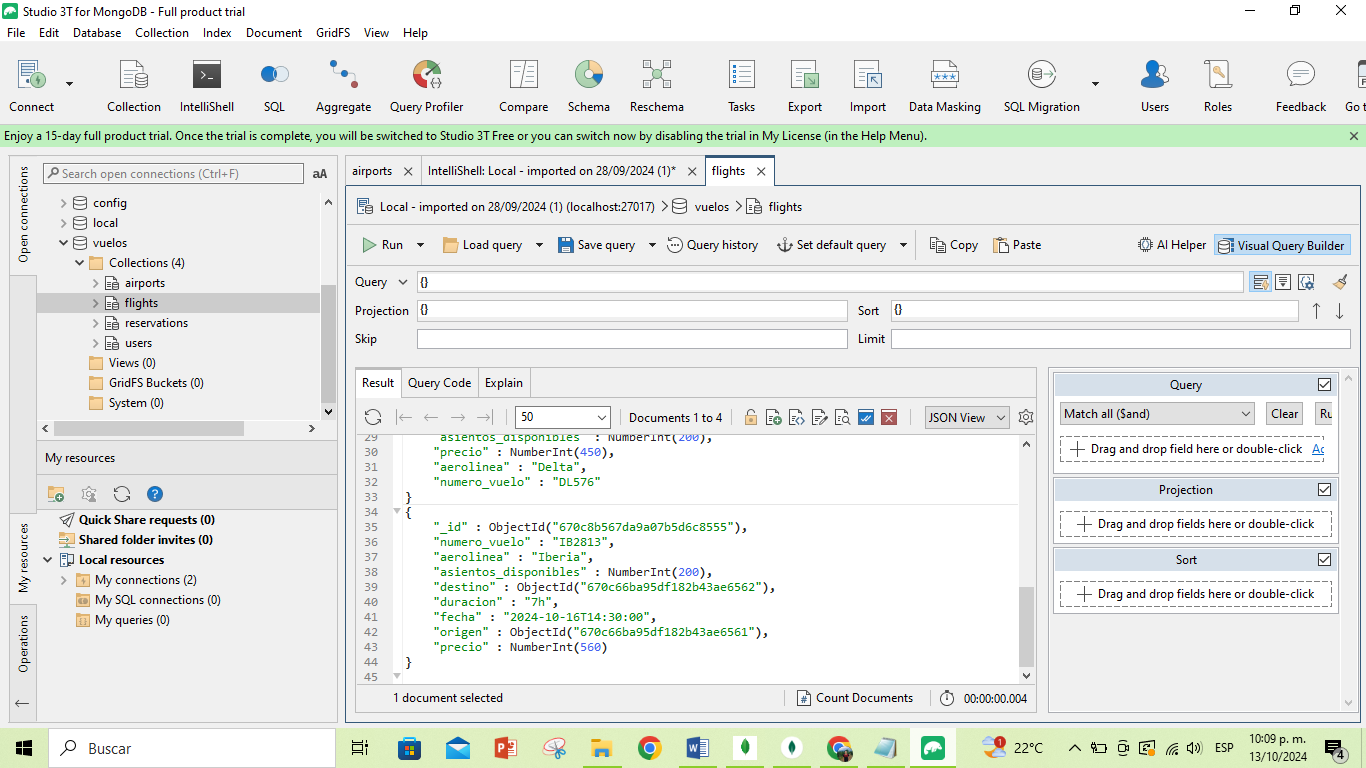
**Nuevas Funciones de Actualización**

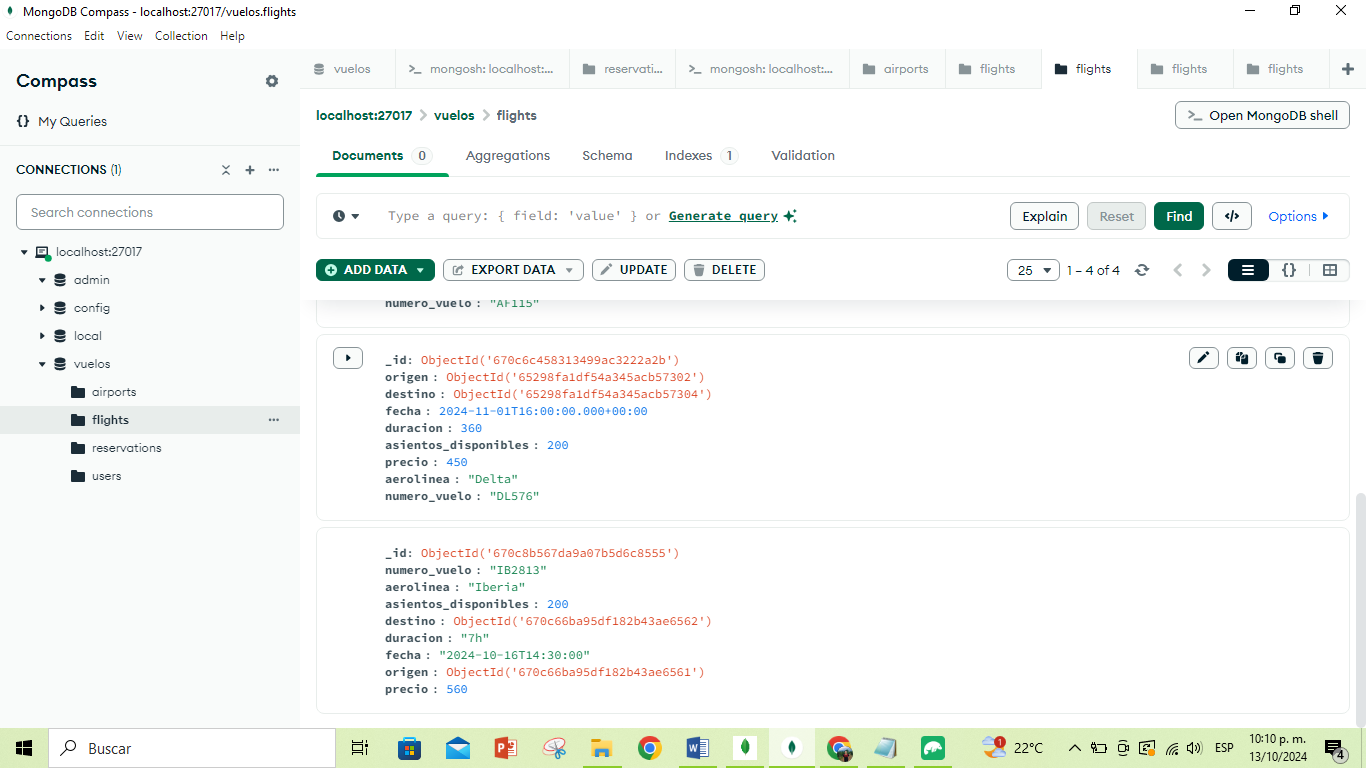
**1. Agregar un nuevo vuelo usando upsert**

Si queremos agregar un nuevo vuelo o actualizar uno existente, podemos hacerlo con **upsert.**

Por ejemplo, aquí estamos insertando un nuevo vuelo

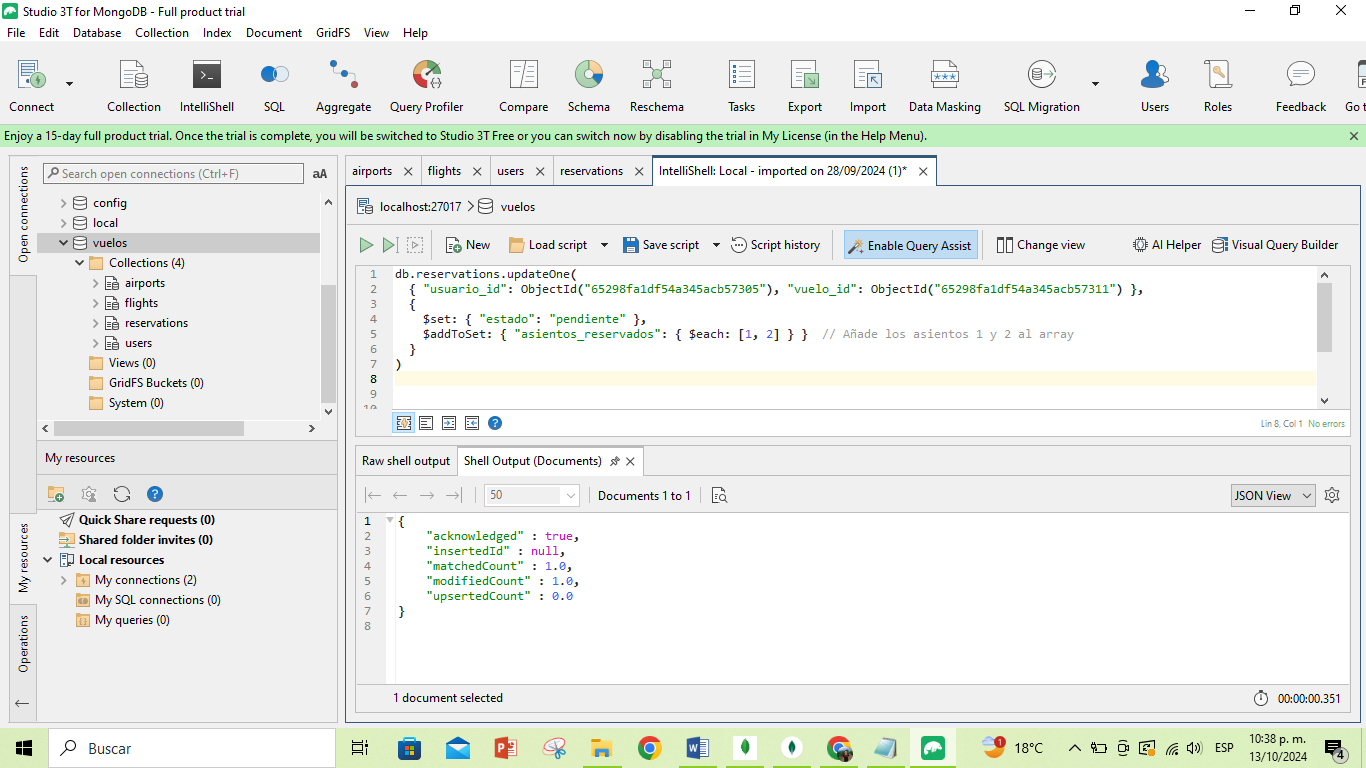


Aquí ya tenemos nuestro nuevo vuelo:

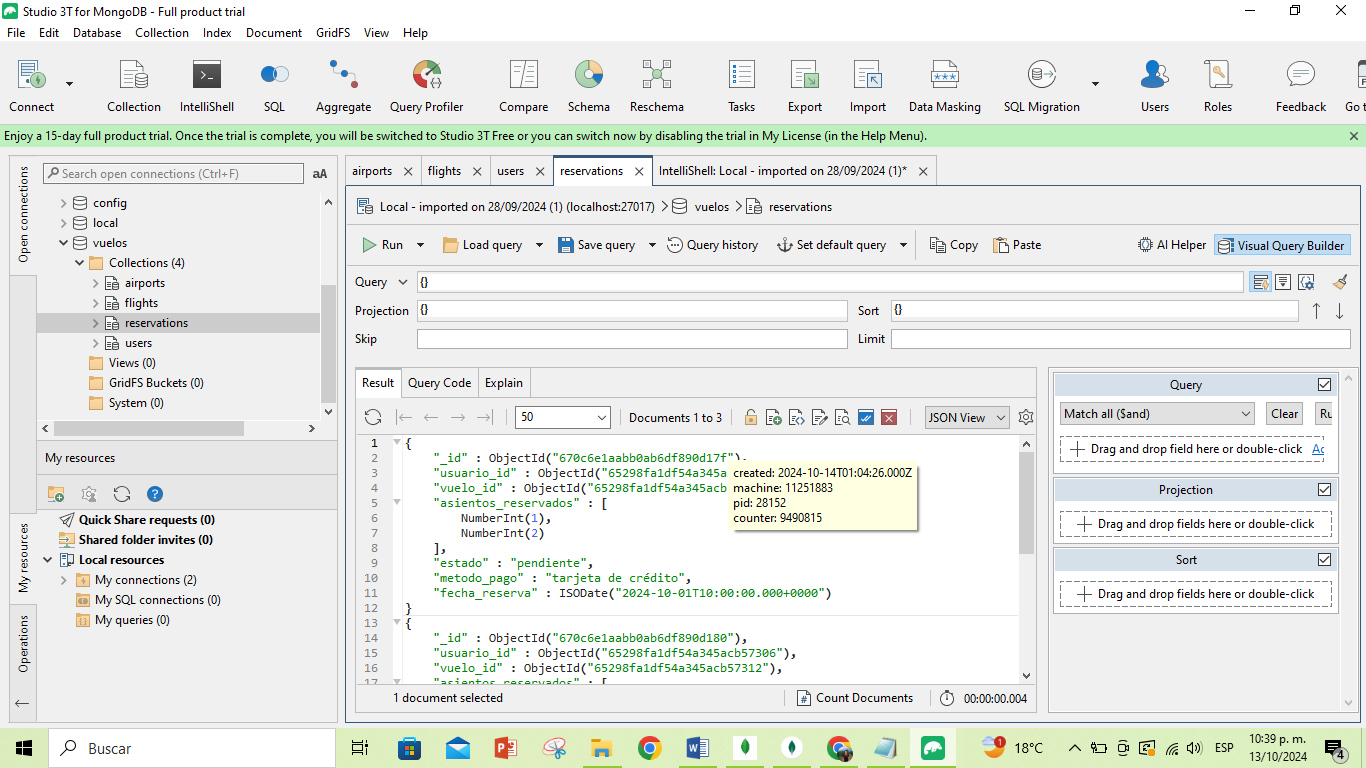


**2. Actualizar las reservas de un usuario usando each**

Si un usuario reserva varios asientos en un vuelo, podemos actualizar la reserva usando $each para agregar múltiples asientos.



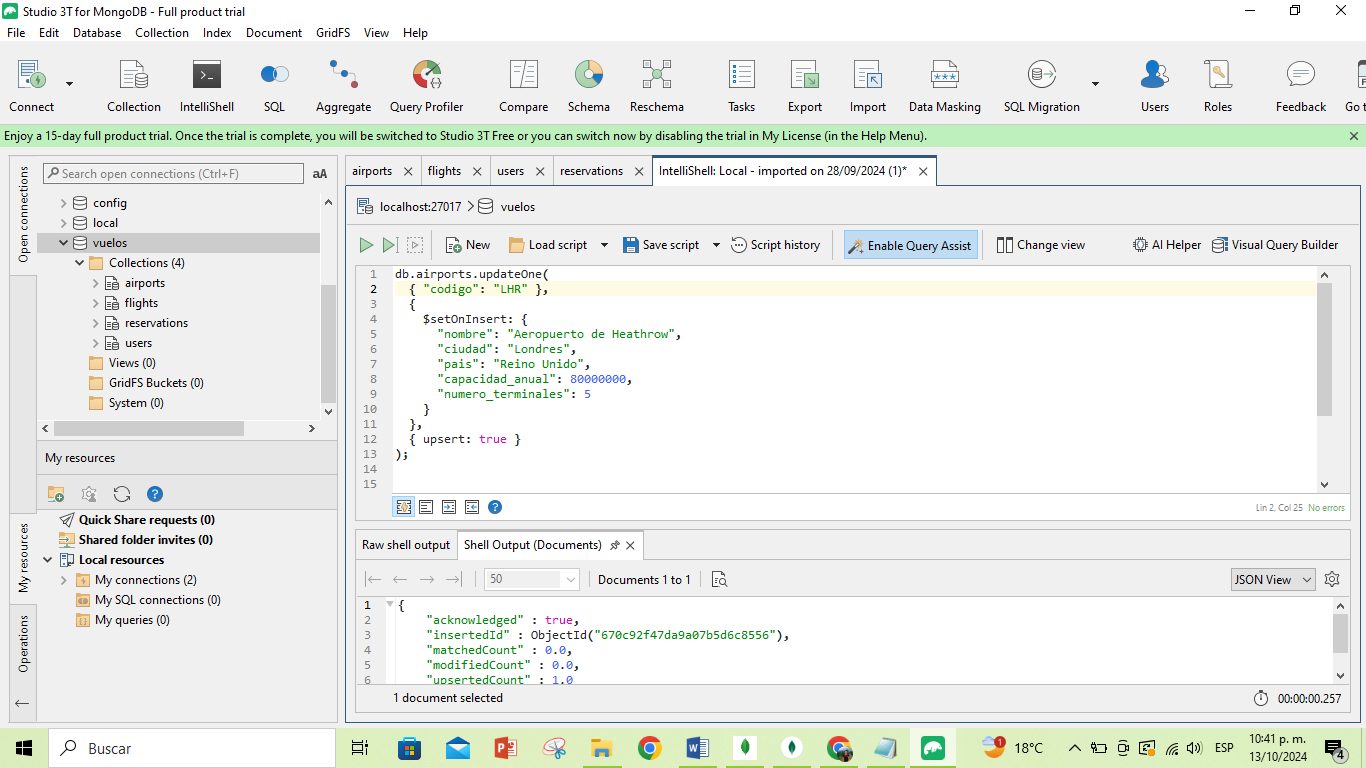
Y aquí tenemos nuestra actualización de reservación de vuelo



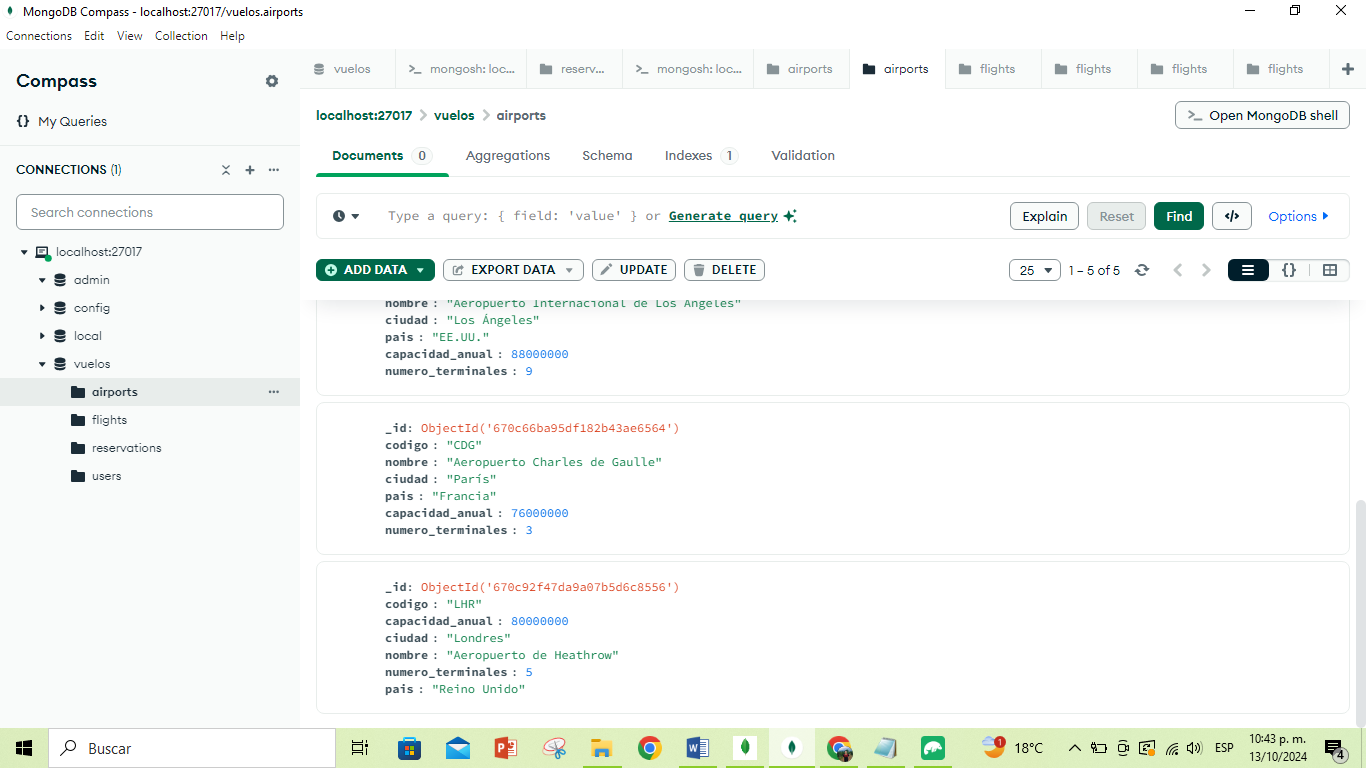
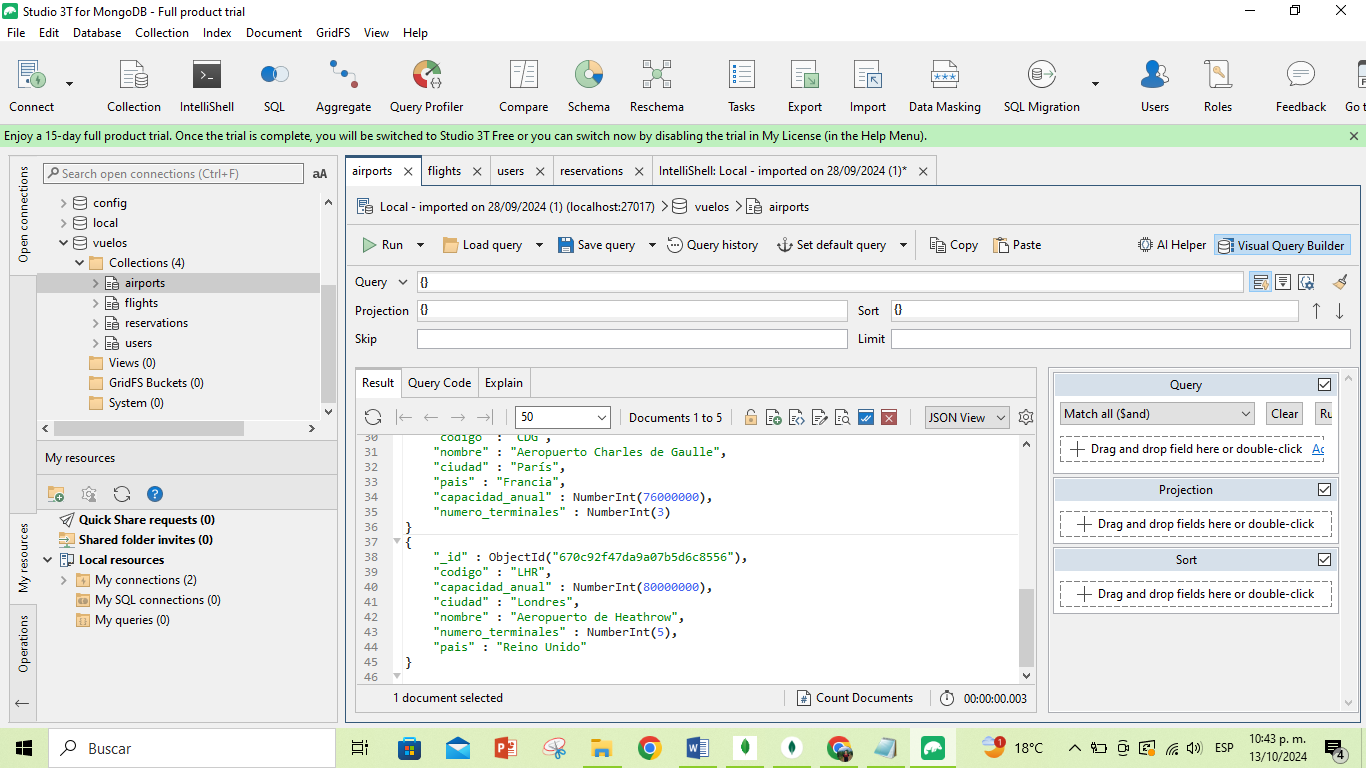
Lo que nos genera aquí es la actualización de vuelos donde dentro de un array podemos obtener muchos vuelos

**3. Actualizar la capacidad de un aeropuerto usando setOnInsert**

Para actualizar la capacidad de un aeropuerto solo si es un nuevo aeropuerto, podemos usar setOnInsert.



Aquí generamos la actualización de un nuevo aeropuerto



Con este diseño ampliado, puedes manejar un sistema más robusto de reservas de vuelos con más datos y funcionalidades realistas. Las relaciones entre colecciones siguen reflejando las necesidades del sistema, y las funciones de actualización permiten mantener la base de datos dinámica y actualizada.

**Relaciones entre colecciones**

**1. Relación uno a uno (1:1)**

* **Usuario - Pasaporte**: Cada usuario tiene un solo pasaporte, que es un atributo del usuario.
  + **Colección involucrada**: users
  + **Estructura**:
    - Un campo pasaporte está dentro de la colección users.

{

"\_id": ObjectId(),

"nombre": "Juan Pérez",

"pasaporte": "AB123456"

}

**2. Relación uno a muchos (1 - m)**

* **Vuelo - Reservas**: Un vuelo puede tener muchas reservas asociadas. Un usuario puede reservar más de un asiento en el mismo vuelo.
  + **Colecciones involucradas**: flights, reservations
  + **Estructura**:
    - Cada documento en la colección reservations contiene referencias a usuario\_id y vuelo\_id.

{

"\_id": ObjectId(),

"origen": ObjectId("..."), // ID del aeropuerto MAD

"destino": ObjectId("..."), // ID del aeropuerto JFK

"fecha": "2024-10-15T14:30:00"

}

**3. Relación muchos a muchos (M-M)**

* **Usuario - Vuelos (a través de Reservas)**: Un usuario puede reservar múltiples vuelos, y un vuelo puede tener múltiples usuarios.
  + **Colecciones involucradas**: users, flights, reservations
  + **Estructura**:
    - La colección reservations sirve de tabla intermedia (relación M-M) que conecta usuarios y vuelos con los campos usuario\_id y vuelo\_id.

{

"\_id": ObjectId(),

"usuario\_id": ObjectId("..."), // ID de Juan Pérez

"vuelo\_id": ObjectId("..."), // ID del vuelo IB2812

"estado": "confirmado"

}

**Resumen de las relaciones:**

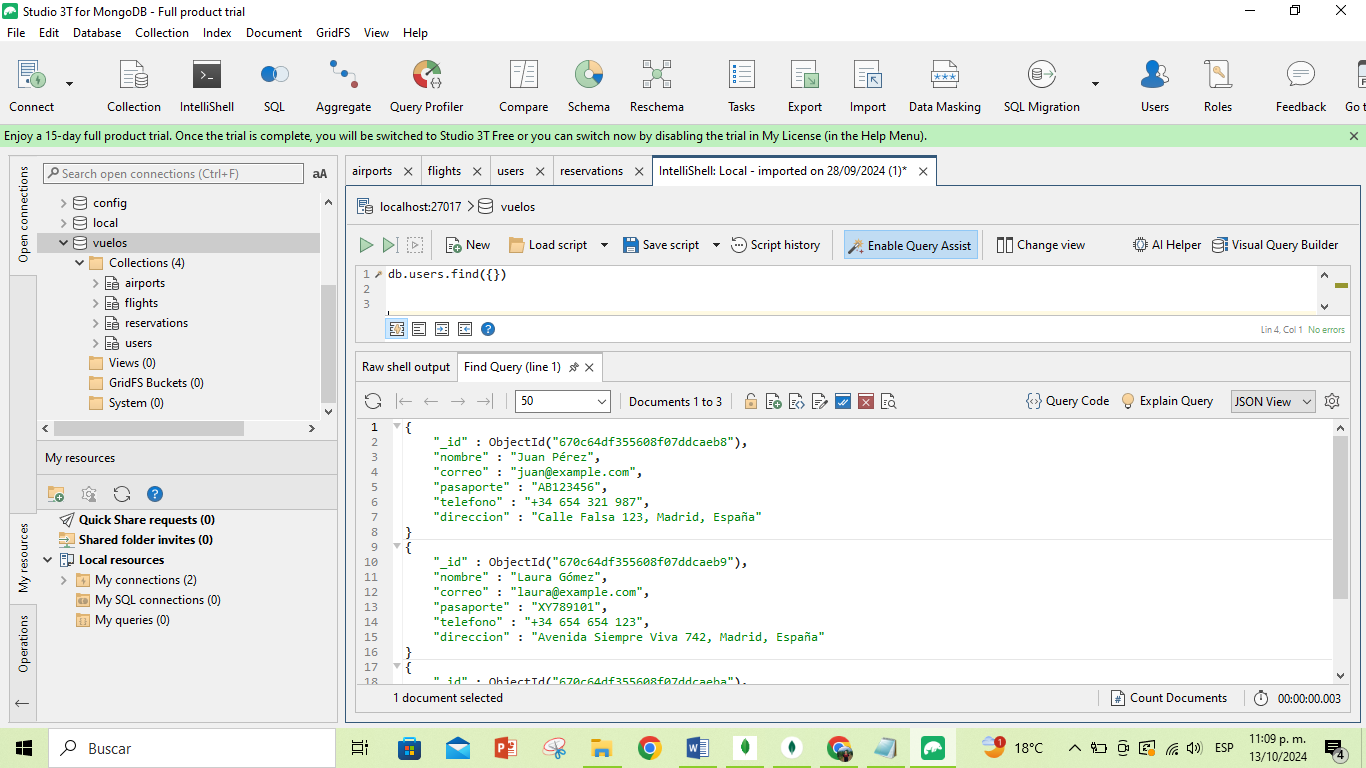
1. **Uno a Uno (1:1)**:
   * Usuario - Pasaporte (Dentro de la colección users)
2. **Uno a Muchos (1-M)**
   * Vuelo - Reservas (Colecciones flights, reservations)
   * Aeropuerto - Vuelos (Colecciones airports, flights)
3. **Muchos a Muchos (M-M)**:
   * Usuario - Vuelos (a través de reservations que conecta users y flights)

Estas relaciones permiten un diseño flexible en MongoDB, permitiendo un fácil escalado y adaptabilidad al sistema de reservas de vuelos.

**CONSULTAS**

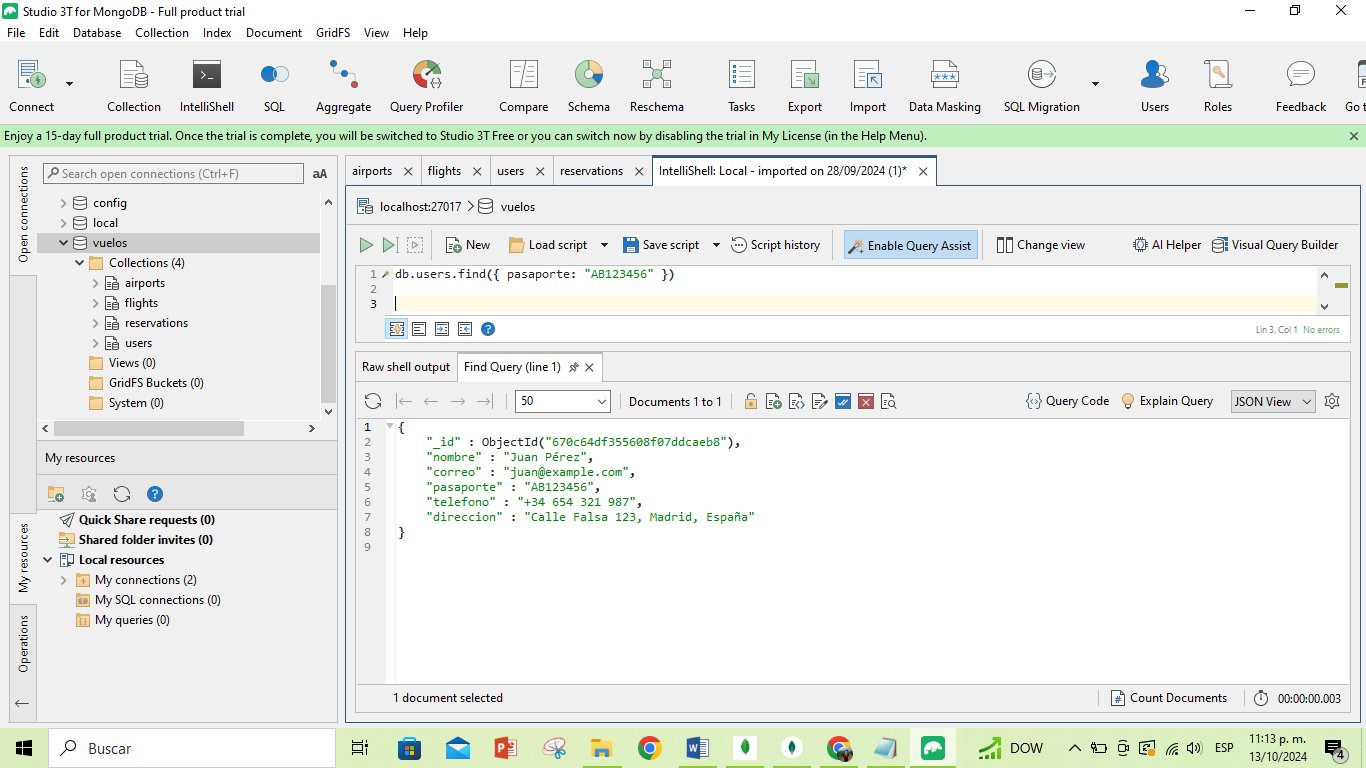
* 1. **Buscar todos los usuarios**

**db.users.find({})**



* 1. **Buscar un usuario por su pasaporte**

**db.users.find({ pasaporte: "AB123456" })**



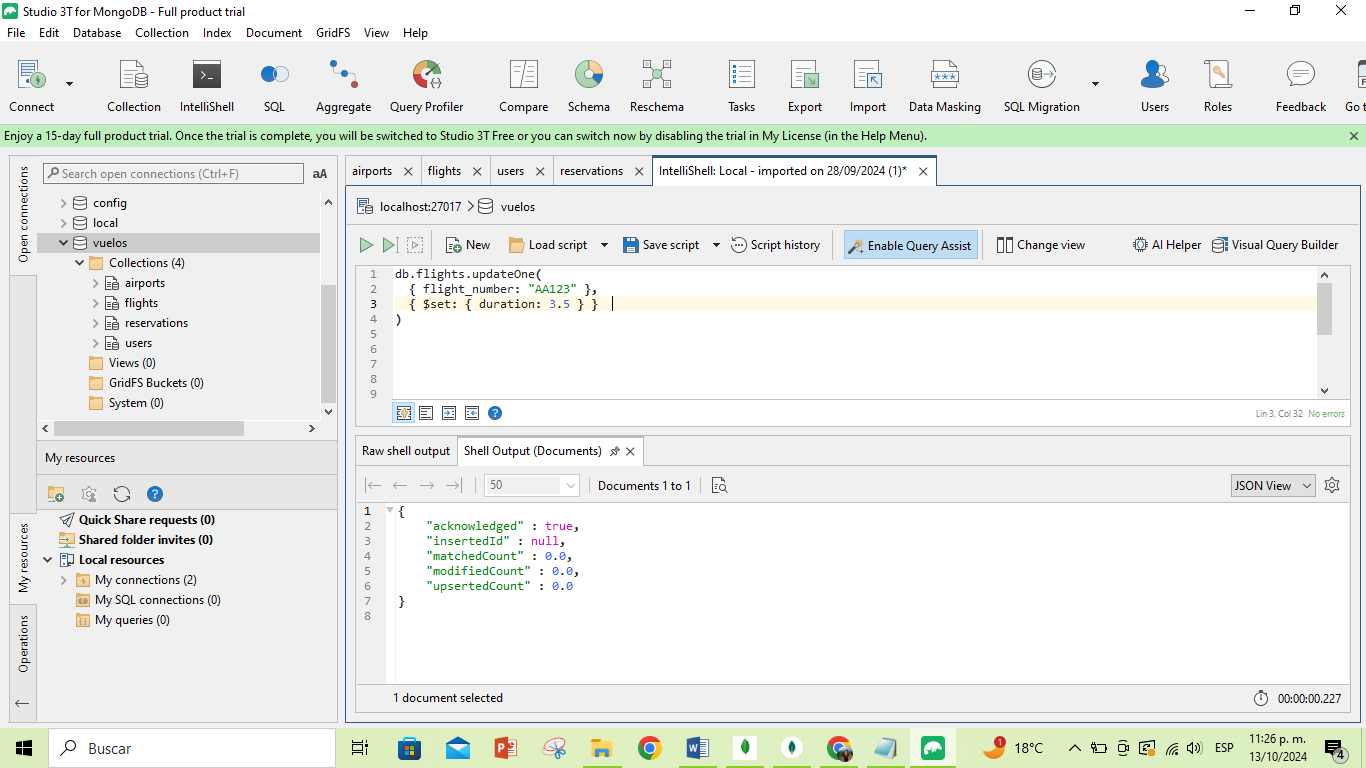
1. Actualizar la duración de un vuelo:

**db.flights.updateOne(**

**{ flight\_number: "AA123" },**

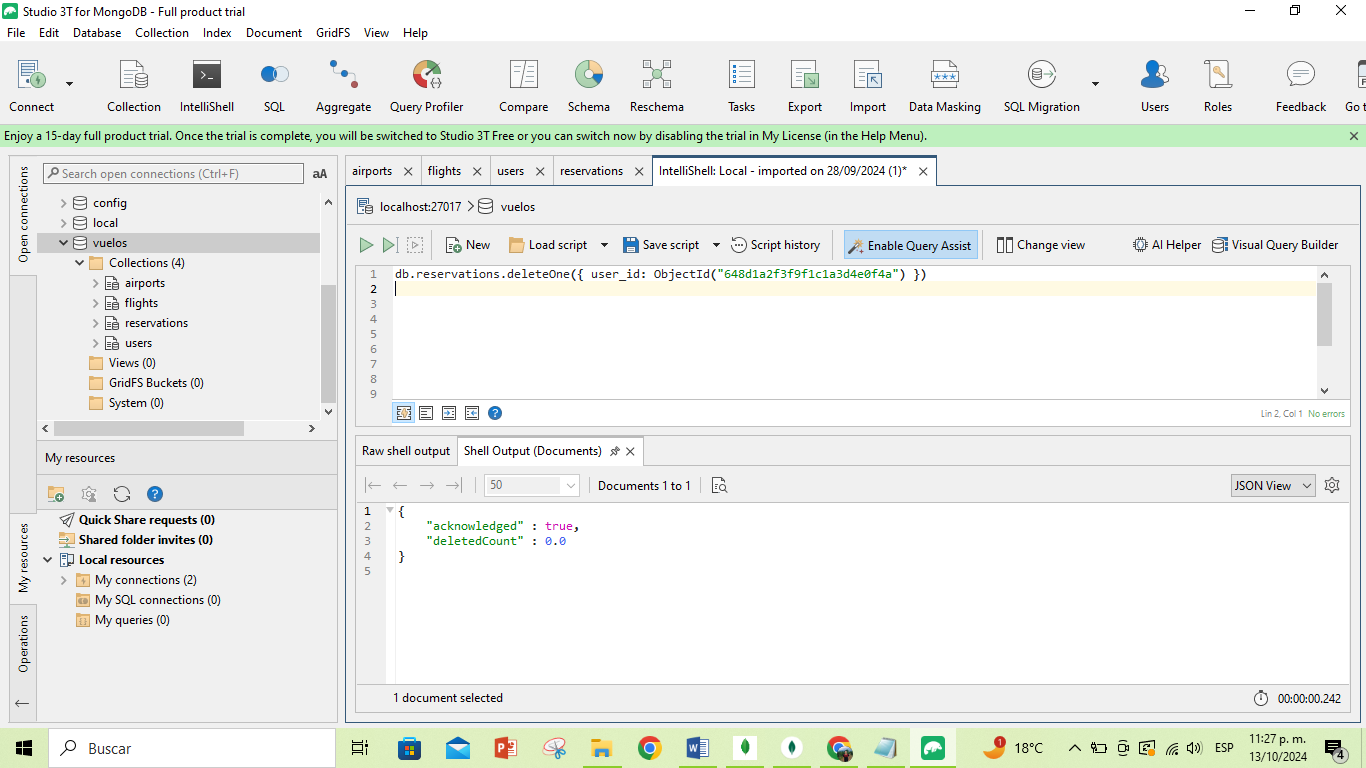
**{ $set: { duration: 3.5 } } // Cambiando la duración a 3.5 horas**

**)**



1. Eliminar una reserva por ID de usuario:

**db.reservations.deleteOne({ user\_id: ObjectId("648d1a2f3f9f1c1a3d4e0f4a") })**



1. Insertar una nueva reserva:

**db.reservations.insertOne({**

**user\_id: ObjectId("648d1a2f3f9f1c1a3d4e0f4a"),**

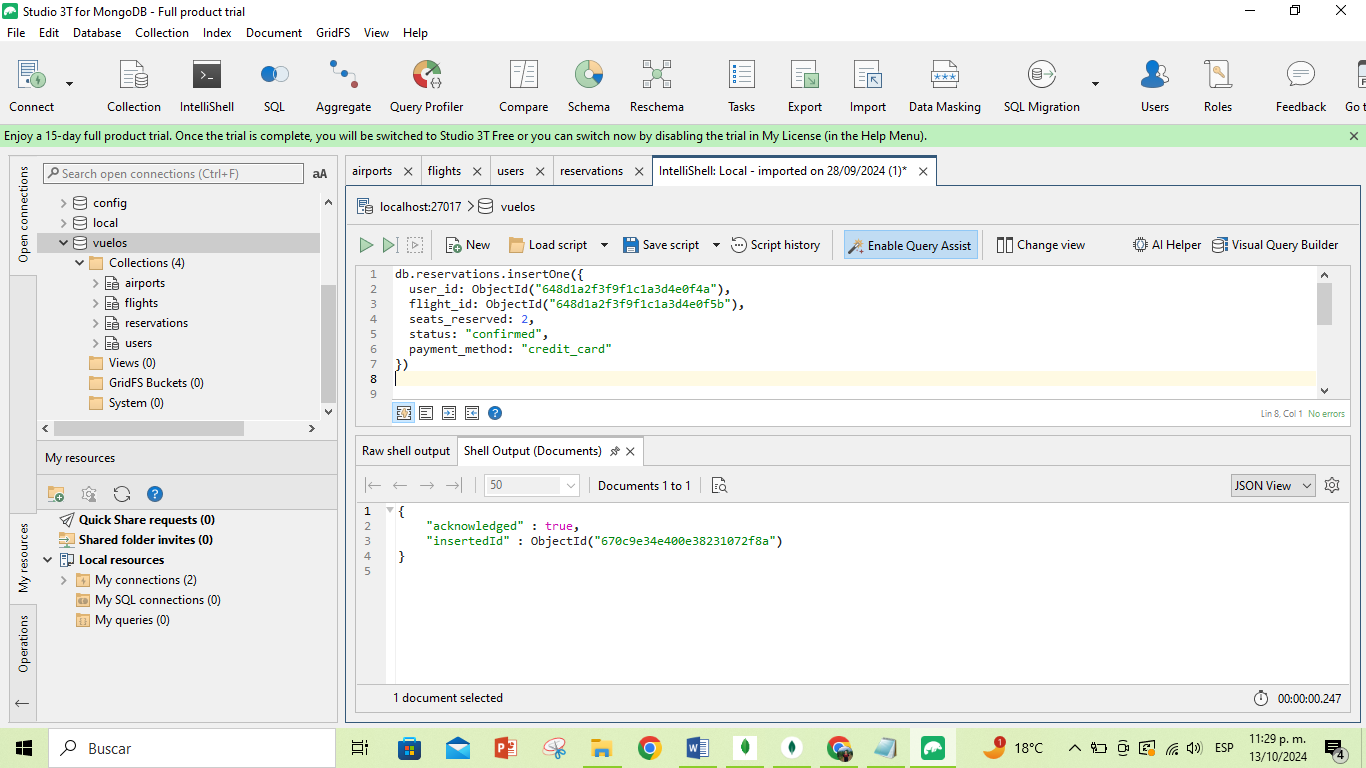
**flight\_id: ObjectId("648d1a2f3f9f1c1a3d4e0f5b"),**

**seats\_reserved: 2,**

**status: "confirmed",**

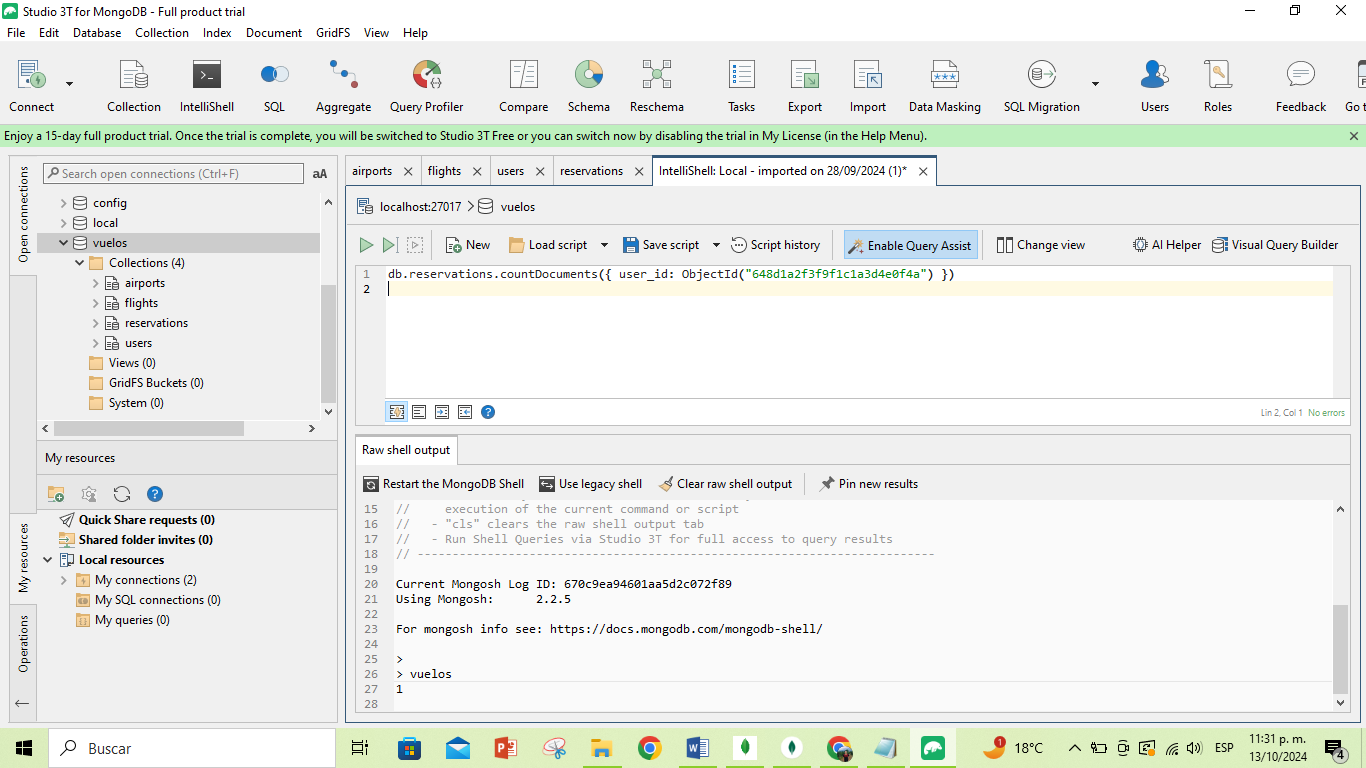
**payment\_method: "credit\_card"**

**})**



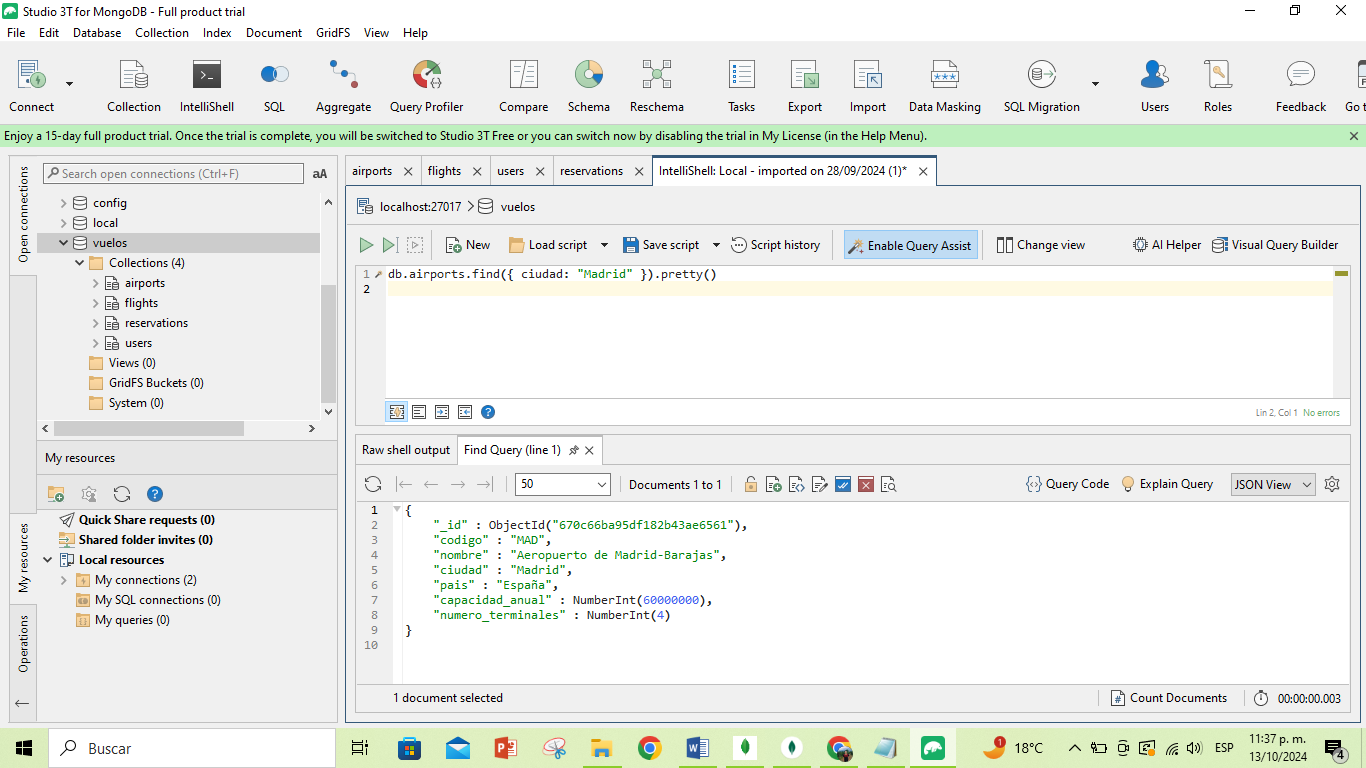
1. Contar cuántas reservas tiene un usuario específico:

**db.reservations.countDocuments({user\_id:ObjectId("648d1a2f3f9f1c1a3d4e0f4a") })**



1. Mostrar todos los aeropuertos en una ciudad específica (ej. paris):

**db.airports.find({ ciudad: "Madrid" }).pretty()**



**DIAGRAMAS DE LAS COLECCIONES**

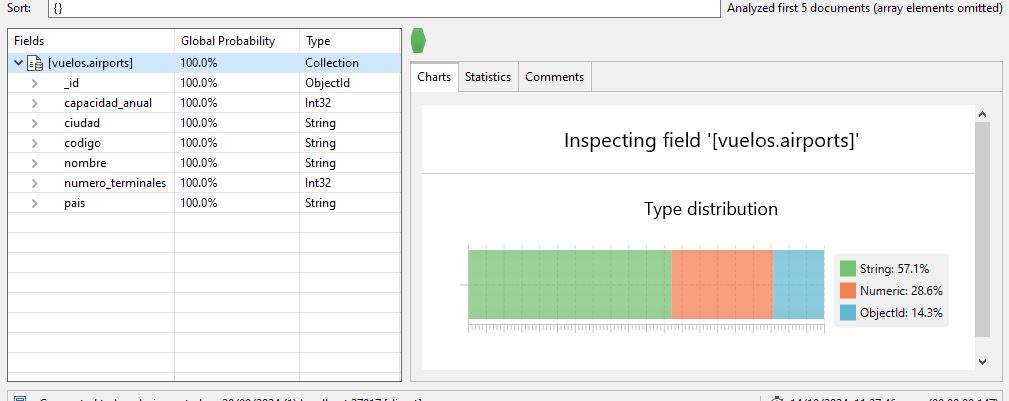
****

Diagrama de la colección airports representa el porcentaje de strings que manejamos en la colección al igual que el porcentaje de datos numéricos y datos tipo objectid.

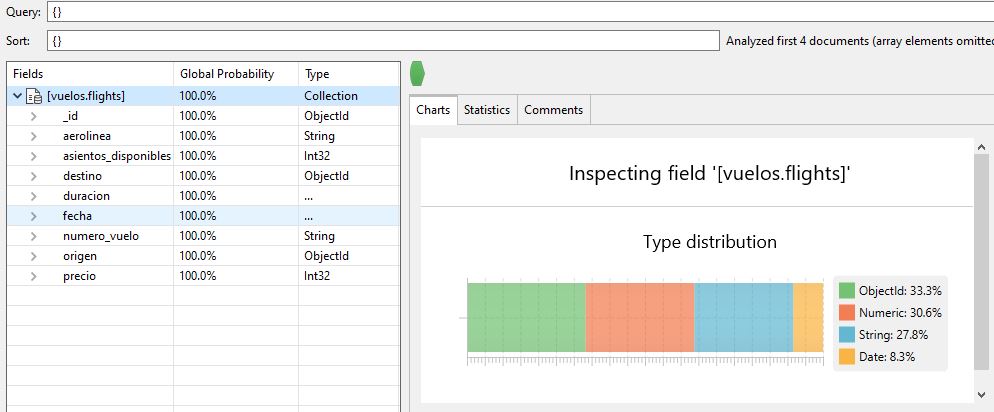
****

Diagrama de la colección flights representa el porcentaje de datos tipo strings que manejamos en la colección al igual que el porcentaje de tipo de datos numéricos, los datos tipo objectid y los tipos de datos de fechas en este caso date.

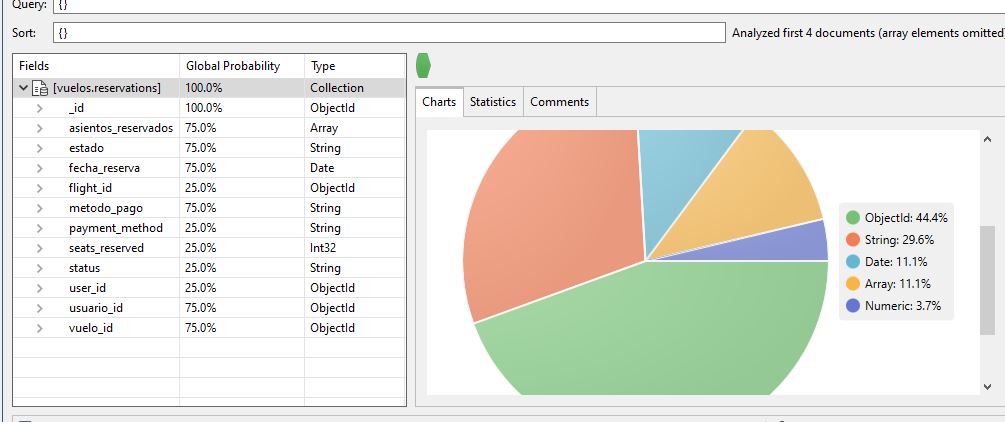
****

Diagrama de la colección reservations representa el porcentaje de datos tipo strings que manejamos en la colección al igual que el porcentaje de tipo de datos numéricos, los datos tipo objectid, los tipos de datos de fechas en este caso date y manejamos un tipo de datos array.

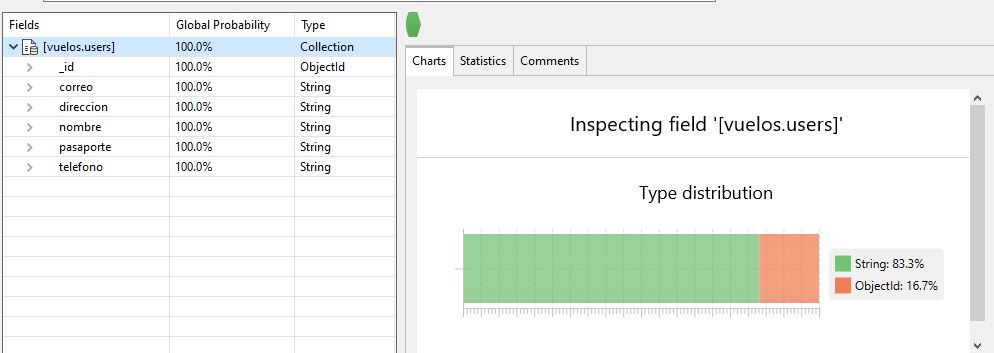
****

Diagrama de la colección users representa el porcentaje de strings que manejamos en la colección al igual que el porcentaje de datos tipo objectid.

#### **Interpretación de Resultados**

Los resultados obtenidos muestran que el sistema diseñado es capaz de manejar grandes volúmenes de datos, con actualizaciones eficientes a través de métodos como upsert, setOnInsert, y la capacidad de manejar relaciones complejas. Las pruebas de consulta realizadas confirmaron que el esquema está bien estructurado para consultas rápidas y actualizaciones simultáneas.

**Conclusiones**

El diseño de la base de datos para el sistema de reservas de vuelos en MongoDB ha demostrado ser eficiente y escalable. Las decisiones de diseño, como el uso de referencias y la normalización de colecciones, han permitido una estructura flexible para manejar la interacción entre usuarios, vuelos, aeropuertos y reservas. Además, el uso de herramientas como Studio 3T y MongoDB Compass facilitó el análisis y la visualización del esquema.

### ****Recomendaciones****

* Utilizar índices en las colecciones para mejorar aún más el rendimiento de las consultas.
* Continuar explorando las capacidades de agregación de MongoDB para generar informes más detallados.
* Considerar la replicación y particionamiento de la base de datos para garantizar alta disponibilidad y rendimiento en un entorno de producción a gran escala.

### ****Referencias****

* MongoDB Documentation: <https://www.mongodb.com/docs/>
* Studio 3T Documentation: <https://studio3t.com/>
* Curso de MongoDB avanzado: <https://www.example.com>
* GitHub documentación: <https://github.com/gaviriaaldeir/CRUD_MONGO>