Universidad de los Andes

Departamento de Ingeniería de Sistemas y Computación

Sistemas transaccionales

Documento de diseño iteración 3 - Proyecto

Profesor

Wilfredy Santamaria Ruiz

Grupo B6

Laura Sofia Murcia – 202123099

Sara Benavides Mora – 202022464

Daniel Alfonso García Pilimur - 202012183

Bogotá - Colombia

Noviembre del 2023

Documento de Análisis y Diseño Entrega 3

Para el desarrollo del proyecto se utilizó el caso Hotel de los Andes, con el objetivo de aprender a modelar correctamente las relaciones que se dan entre los clientes, servicios y otros autores que comprende el mundo del problema. Todo esto con el propósito de afianzar habilidades de modelado de datos no relacionales a partir del enunciado propuesto. Adicionalmente, se afianzaron habilidades dentro del diseño e implementación de MongoDB como base de datos.

Introducción

En este informe se encuentra toda la información pertinente para entender el funcionamiento de la aplicación realizada y sus funciones. El propósito principal de la aplicación construida es facilitar el procesamiento de los datos recibidos por el hotel para hacer más efectivo los procesos de recolección de información. Esto se hace por medio de la implementación y diseño de una base de datos no relacional usando la estrategia de embeber la información.

Arquitectura de la Aplicación

La aplicación construida se realizó mediante la base de datos MongoDB Compass versión 1.40.4. Se utilizó Java 17 y Maven para el desarrollo de la aplicación.

El diseño de esta aplicación se centra en que el usuario pueda interactuar con la base de datos por medio de una interfaz html. Esto le permitirá al usuario un manejo fácil y práctico de los datos y las consultas pertinentes sobre ellos.

Restricciones del proyecto/Reglas de dominio

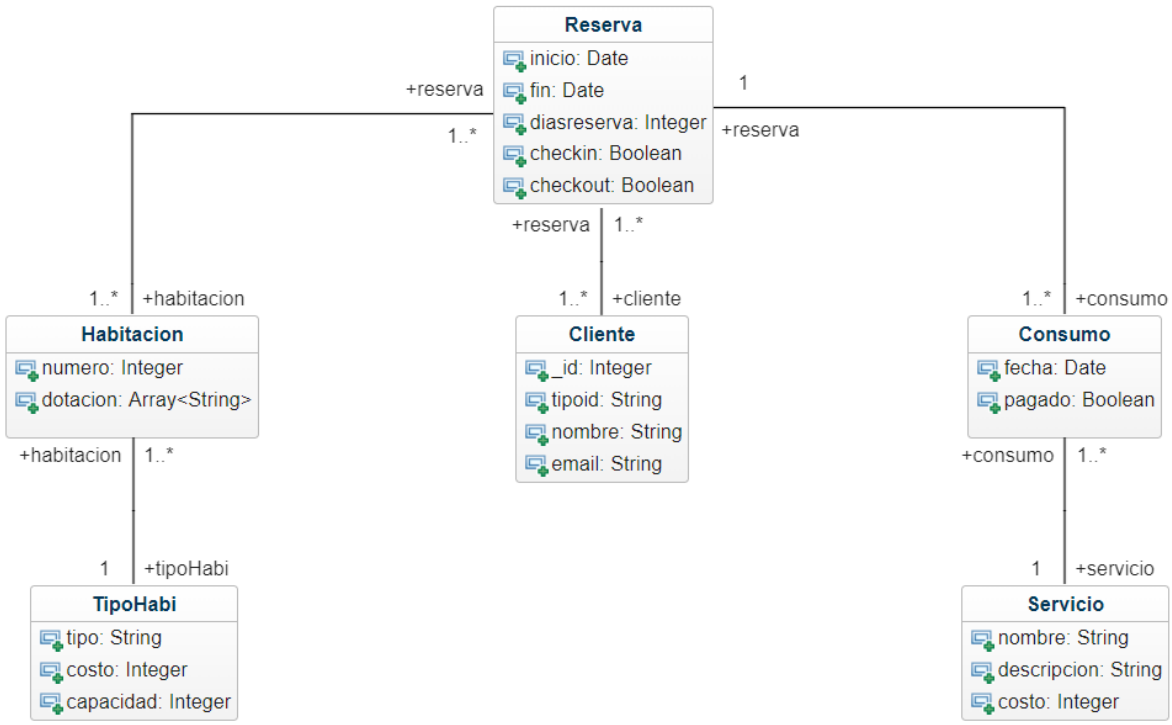
1. El costo de alojamiento se define por noche.
2. Ya no hay manejo de tipos de usuarios.
3. Cada hotel cuenta con servicios en los que se pueden incluir: piscina, gimnasio, internet, bares, restaurantes, supermercado, tiendas, SPA, lavandería, préstamo de utensilios, salones de reuniones y salones de conferencia.
4. Se debe poder gestionar la entrada y salida de los clientes de una reserva.
5. Los servicios ya no necesitan reservas para ser consumidos.
6. Los tipos de habitaciones incluyen: suite presidencial, suite, familiar, doble y sencilla.

Diseño de la Base de Datos

Para el diseño de la base de datos de la aplicación, se generó un diagrama UML que permitiera generar una abstracción del problema general. Este se modeló haciendo uso de la información brindada en el enunciado donde se identificaron las restricciones y los atributos de las entidades a modelar para la implementación de la aplicación que se quiere desarrollar. A continuación, se pueden ver el diagrama mencionado.

Diagrama UML (desarrollado en genmymodel)

En este diagrama se encuentran todas las entidades identificadas junto con sus respectivas relaciones. En total, se identificaron 6 entidades con las cuales se modeló el problema.



Análisis de la carga de trabajo (workload)

Teniendo en cuenta el modelo conceptual UML presentado en la sección anterior, se identificaron las siguientes entidades con sus respectivos atributos:

1. **Reserva**: (Date inicio, Date fin, Integer diasreserva, Boolean checkin, Boolean checkout)
2. **Habitación**: (Integer número, Array<String> dotación)
3. **TipoHabi**: (String tipo, Integer costo, Integer capacidad)
4. **Cliente**: (Integer _id, String tipoid, String nombre, String email)
5. **Consumo**: (Date fecha, Boolean pagado)
6. **Servicio**: (String nombre, String descripción, Integer costo)

Ahora bien, con esta información se procede a cuantificar la cantidad de registros que tendría la base de datos para cada una de las entidades. Para ello se tuvo en cuenta el aproximado mencionado en el enunciado y teniendo en cuenta nuestro diseño de la BD se llegó a el siguiente estimado:

Se espera que, para un hotel de gran magnitud, la cantidad de registros para los tipos de habitación podrían llegar a ser entre 15 y 22, las habitaciones a las 250 – 300 y los servicios a 25 – 30. En cuanto a las reservas de una habitación, se estima que por año se puede llegar a tener hasta 20.000, lo cual quiere decir que en una ventana de 3 años se podría llegar a

60.000 reservaciones. Finalmente, para los consumos de servicios, se espera que sea cerca de 300.000 en una ventana de 3 años.

Como esta es una estimación a futuro, para esta entrega se tendrán en cuenta menor cantidad de registros por entidad puesto que con eso es suficiente para mostrar el funcionamiento de la aplicación propuesta y la forma en la que esta está trabajando. Se manejarán entonces 4 tipos de habitaciones, 31 habitaciones y 10 servicios que presta el hotel. Con esta información se ingresaron entonces 102 reservas realizadas por 101 usuarios los cuales realizaron un total de 298 consumos en total.

A continuación, se presenta el análisis de las operaciones de lectura y escritura para cada entidad:

Anexo A

Entities	Operations	Information Needed	Type
Tipos de habitaciones	Crear/modificar los tipos de habitaciones	Detalle del tipo de habitación	Escribe
Tipos de habitaciones	Consulta de un tipo de habitación	Detalle del tipo de habitación + reserva	Lee
Habitaciones	Crear/modificar una habitación	Detalle de la habitación	Escribe
Habitaciones	Consulta de una habitación	Detalle de la habitación + reserva	Lee
Servicios	Crear/modificar un servicio	Detalle del servicio	Escribe
Servicios	Consultar un servicio	Detalle del servicio + consumo + reserva	Lee
Reservas	Crear/modificar una reserva de una habitación	Detalle de la reserva + detalle de la habitación + detalle del tipo de habitación	Escribe
Reservas	Consulta de una reserva	Detalle de la reserva	Lee
Consumos	Crear/modificar un consumo hecho por un cliente	Detalle del consumo + detalle del servicio + detalle del cliente + detalle de la reserva	Escribe
Consumos	Consulta de un consumo hecho por un cliente	Detalle de la reserva + detalle del consumo + detalle del cliente	Lee
Reserva	Crear/modificar la llegada y partida de un cliente	Detalle de la reserva + detalle del cliente + detalle de los consumos	Escribe
Reserva	Consulta de la llegada y partida de un cliente	Detalle de la reserva + detalle del cliente	Lee

Ahora, se presenta la cuantificación de las operaciones de lectura y escritura para cada entidad:

Anexo B

Entities	Operation	Information Needed	Type	Rate
Tipos de habitaciones	Crear/modificar los tipos de habitaciones	Detalle del tipo de habitación	Escribe	1/semana
Tipos de habitaciones	Consulta de un tipo de habitación	Detalle del tipo de habitación + reserva	Lee	1/semana
Habitaciones	Crear/modificar una habitación	Detalle de la habitación	Escribe	2/semana
Habitaciones	Consulta de una habitación	Detalle de la habitación + reserva	Lee	2/semana
Servicios	Crear/modificar un servicio	Detalle del servicio	Escribe	2/semana
Servicios	Consultar un servicio	Detalle del servicio + consumo + reserva	Lee	2/semana
Reservas	Crear/modificar una reserva de una habitación	Detalle de la reserva + detalle de la habitación + detalle del tipo de habitación	Escribe	100/día
Reservas	Consulta de una reserva	Detalle de la reserva	Lee	50/día
Consumos	Crear/modificar un consumo hecho por un cliente	Detalle del consumo + detalle del servicio + detalle del cliente + detalle de la reserva	Escribe	500/día
Consumos	Consulta de un consumo hecho por un cliente	Detalle de la reserva + detalle del consumo + detalle del cliente	Lee	250/día
Reserva	Crear/modificar la llegada y partida de un cliente	Detalle de la reserva + detalle del cliente + detalle de los consumos	Escribe	100/día
Reserva	Consulta de la llegada y partida de un cliente	Detalle de la reserva + detalle del cliente	Lee	50/día

Descripción de las entidades y sus relaciones

Para poder llegar al modelo final de nuestra base de datos se realizó un análisis exhaustivo sobre las entidades encontradas junto con las relaciones que existen entre ellas, así que, a continuación, se presenta la lista de entidades con su respectiva descripción.

1. **Reserva**: almacena las reservas de las habitaciones del hotel con sus respectivas fechas y clientes que hacen parte de la reserva.
2. **Habitación**: guarda la información de una habitación tales como su numero como identificador único y la dotación que esta tiene. Además, también muestra cual es el tipo de habitación que posee.
3. **TipoHabi**: almacena la información de los tipos de habitaciones existentes en el hotel.

4. **Cliente**: muestra la información de los clientes que tenido o tienen reservas en el hotel.
5. **Consumo**: guarda los consumos realizados por los clientes de una reserva durante su estadía junto con el precio, si ya está pagado y la fecha en la que el consumo se realizó.
6. **Servicio**: almacena la información de los servicios que presta el hotel junto con el precio que este tiene y su descripción.

Ahora, se procede a realizar el respectivo análisis detallado de las relaciones que hay entre cada una de las entidades haciendo especial énfasis en su cardinalidad. Con lo anterior se seleccionará el esquema de asociación para cada relación, ya sea embebido o referenciado, mediante el análisis del esquema de relación entre entidades y usando los resultados encontrados luego de haber realizado el análisis de la carga de trabajo presentado anteriormente.

1. **Reserva – Habitación (Muchos a muchos)**

Guideline Name	Question	Embed	Reference
Simplicity	Would keeping the pieces of information together lead to a simpler data model and code?	Yes	No
Go together	Do the pieces of information have a “has-a”, “contains”, or similar relationship?	Yes	No
Query atomicity	Does the application query the pieces of information together?	Yes	No
Update complexity	Are the pieces of information updated together?	No	Yes
Archival	Should the pieces of information be archived at the same time?	Yes	No
Cardinality	Is there a high cardinality (current or growing) in the child side of the relationship?	Yes	No
Data duplication	Would data duplication be too complicated to manage and undesired?	No	Yes
Document size	Would the combined size of the pieces of information take too much memory or transfer bandwidth for the application?	No	Yes
Document growth	Would the embedded piece grow without bound?	No	Yes
Workload	Are the pieces of information written at different times in a write-heavy workload?	No	Yes
Individuality	For the children side of the relationship, can the pieces exist by themselves without a parent?	No	Yes
TOTAL		9	2

2. **Habitación – TipoHabi (Uno a muchos)**

Guideline Name	Question	Embed	Reference
Simplicity	Would keeping the pieces of information together lead to a simpler data model and code?	Yes	No
Go together	Do the pieces of information have a “has-a”, “contains”, or similar relationship?	Yes	No
Query atomicity	Does the application query the pieces of information together?	Yes	No
Update complexity	Are the pieces of information updated together?	No	Yes
Archival	Should the pieces of information be archived at the same time?	Yes	No
Cardinality	Is there a high cardinality (current or growing) in the child side of the relationship?	No	Yes
Data duplication	Would data duplication be too complicated to manage and undesired?	Yes	No
Document size	Would the combined size of the pieces of information take too much memory or transfer bandwidth for the application?	No	Yes
Document growth	Would the embedded piece grow without bound?	No	Yes
Workload	Are the pieces of information written at different times in a write-heavy workload?	No	Yes
Individuality	For the children side of the relationship, can the pieces exist by themselves without a parent?	Yes	No
TOTAL		8	3

3. Reserva – Cliente (Muchos a muchos)

Guideline Name	Question	Embed	Reference
Simplicity	Would keeping the pieces of information together lead to a simpler data model and code?	Yes	No
Go together	Do the pieces of information have a “has-a”, “contains”, or similar relationship?	Yes	No
Query atomicity	Does the application query the pieces of information together?	Yes	No
Update complexity	Are the pieces of information updated together?	Yes	No
Archival	Should the pieces of information be archived at the same time?	Yes	No

Cardinality	Is there a high cardinality (current or growing) in the child side of the relationship?	No	Yes
Data duplication	Would data duplication be too complicated to manage and undesired?	No	Yes
Document size	Would the combined size of the pieces of information take too much memory or transfer bandwidth for the application?	No	Yes
Document growth	Would the embedded piece grow without bound?	No	Yes
Workload	Are the pieces of information written at different times in a write-heavy workload?	No	Yes
Individuality	For the children side of the relationship, can the pieces exist by themselves without a parent?	No	Yes
TOTAL		11	0

4. Reserva – Consumo (Uno a muchos)

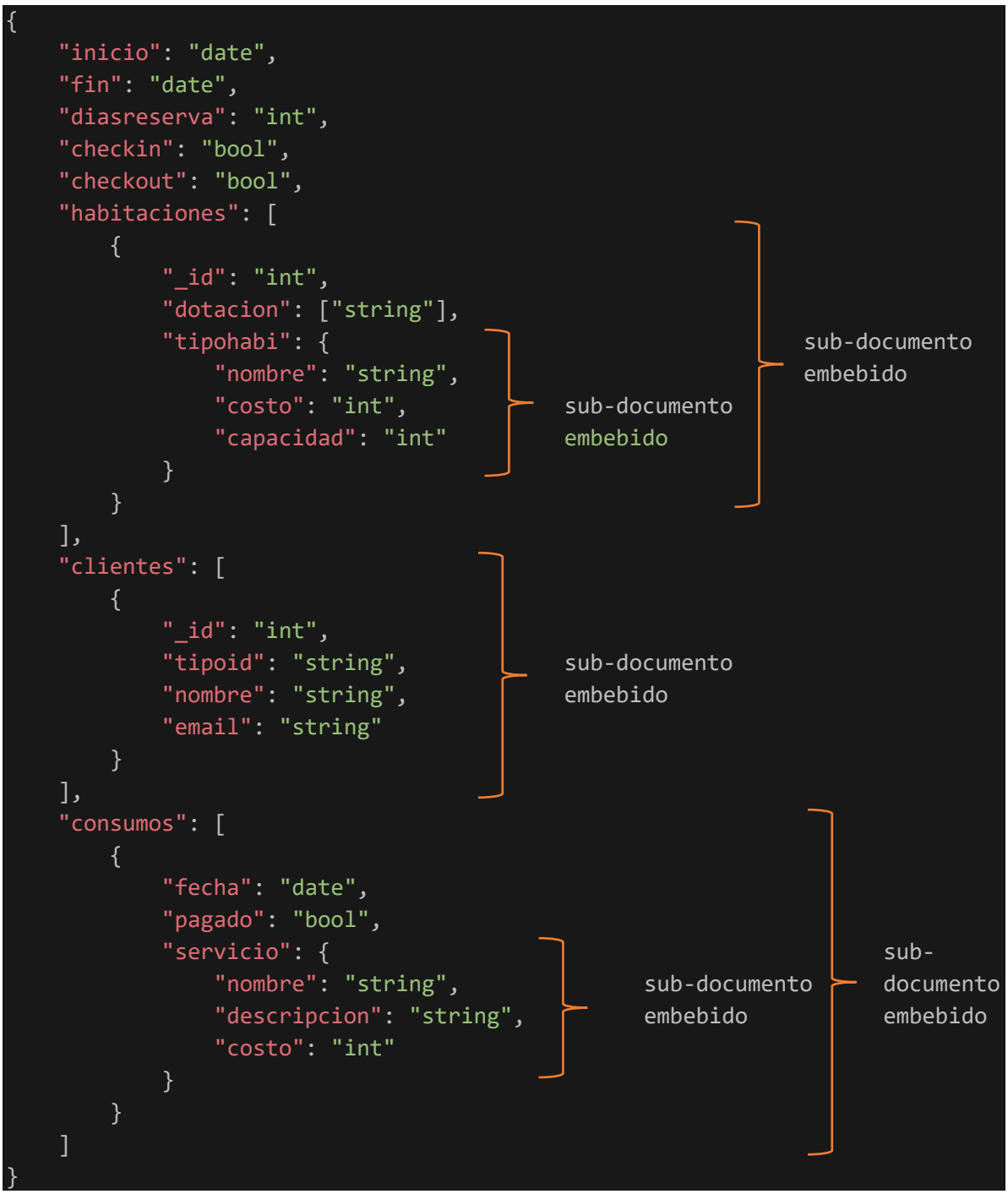
Guideline Name	Question	Embed	Reference
Simplicity	Would keeping the pieces of information together lead to a simpler data model and code?	Yes	No
Go together	Do the pieces of information have a “has-a”, “contains”, or similar relationship?	Yes	No
Query atomicity	Does the application query the pieces of information together?	Yes	No
Update complexity	Are the pieces of information updated together?	Yes	No
Archival	Should the pieces of information be archived at the same time?	Yes	No
Cardinality	Is there a high cardinality (current or growing) in the child side of the relationship?	Yes	No
Data duplication	Would data duplication be too complicated to manage and undesired?	No	Yes
Document size	Would the combined size of the pieces of information take too much memory or transfer bandwidth for the application?	No	Yes
Document growth	Would the embedded piece grow without bound?	No	Yes
Workload	Are the pieces of information written at different times in a write-heavy workload?	Yes	No
Individuality	For the children side of the relationship, can the pieces exist by themselves without a parent?	No	Yes
TOTAL		9	2

5. **Consumo – Servicio (Uno a muchos)**

Guideline Name	Question	Embed	Reference
Simplicity	Would keeping the pieces of information together lead to a simpler data model and code?	Yes	No
Go together	Do the pieces of information have a “has-a”, “contains”, or similar relationship?	Yes	No
Query atomicity	Does the application query the pieces of information together?	Yes	No
Update complexity	Are the pieces of information updated together?	No	Yes
Archival	Should the pieces of information be archived at the same time?	Yes	No
Cardinality	Is there a high cardinality (current or growing) in the child side of the relationship?	No	Yes
Data duplication	Would data duplication be too complicated to manage and undesired?	No	Yes
Document size	Would the combined size of the pieces of information take too much memory or transfer bandwidth for the application?	No	Yes
Document growth	Would the embedded piece grow without bound?	No	Yes
Workload	Are the pieces of information written at different times in a write-heavy workload?	Yes	No
Individuality	For the children side of the relationship, can the pieces exist by themselves without a parent?	Yes	No
TOTAL		8	3

Teniendo en cuenta este análisis, se llegó a la conclusión de que para todas las relaciones del modelo se trabajara sobre un esquema embebido puesto que, si bien hay una que otra razón para manejar el sistema de referenciación, son más las razones para que este sea embebido, tal y como se muestra en el total de puntos recolectados en cada tabla. Esto facilitará el cumplimiento de requerimientos funcionales y no funcionales requeridos por la aplicación y también permitirá que el acceso a la información sea más eficiente.

Para entender de mejor forma la estructura final que va a tener nuestra base de datos se va a crear una colección llamada “hotel” la cual almacenará documentos de las reservas que tiene o ha tenido el hotel. Dentro de cada reserva se van a encontrar sus atributos junto con el resto de la información embebida con la siguiente estructura:



Para finalizar, se creó el respectivo esquema de validación para la colección haciendo uso de MongoDB Compass para verificar que cuando se cree una nueva reserva, esta si o sí tenga todos sus atributos de forma obligatoria a excepción de la parte de consumos puesto que puede que aún no se haya realizado ningún consumo por algún cliente, pero de resto una reserva no puede existir si no tiene al menos una habitación con su tipo de habitación, un

cliente, una fecha de inicio y fin, la cantidad de días que va a tener esa reserva y el estado actual del checkin y checkout. Con esto en mente se aplicó el siguiente validador:

```
{
    $jsonSchema: {
      bsonType: 'object',
      required: [
        'inicio',
        'fin',
        'diasreserva',
        'checkin',
        'checkout',
        'habitaciones',
        'clientes'
      ],
      properties: {
        _id: {
          bsonType: 'objectId'
        },
        inicio: {
          bsonType: 'date'
        },
        fin: {
          bsonType: 'date'
        },
        diasreserva: {
          bsonType: 'int'
        },
        checkin: {
          bsonType: 'bool'
        },
        checkout: {
          bsonType: 'bool'
        },
        habitaciones: {
          bsonType: 'array',
          items: {
            bsonType: 'object',
            required: [
              'numero',
              'dotacion',
              'tipohabi'
            ],
            properties: {
              _id: {
```

```
            bsonType: 'objectId'
          },
          numero: {
            bsonType: 'int'
          },
          dotacion: {
            bsonType: 'array',
            items: {
              bsonType: 'string'
            }
          },
          tipohabi: {
            bsonType: 'object',
            required: [
              'nombre',
              'costo',
              'capacidad'
            ],
            properties: {
              _id: {
                bsonType: 'objectId'
              },
              nombre: {
                bsonType: 'string'
              },
              costo: {
                bsonType: 'int'
              },
              capacidad: {
                bsonType: 'int'
              }
            }
          }
        }
      },
      clientes: {
        bsonType: 'array',
        items: {
          bsonType: 'object',
          required: [
            '_id',
            'tipoid',
            'nombre',
            'email'
```

```
        ],
        properties: {
          _id: {
            bsonType: 'int'
          },
          tipoid: {
            bsonType: 'string'
          },
          nombre: {
            bsonType: 'string'
          },
          email: {
            bsonType: 'string'
          }
        }
      }
    },
    consumos: {
      bsonType: 'array',
      items: {
        bsonType: 'object',
        required: [
          'fecha',
          'pagado',
          'servicio'
        ],
        properties: {
          _id: {
            bsonType: 'objectId'
          },
          fecha: {
            bsonType: 'date'
          },
          pagado: {
            bsonType: 'bool'
          },
          servicio: {
            bsonType: 'object',
            required: [
              'nombre',
              'descripcion',
              'costo'
            ],
            properties: {
              _id: {
```

```
                  bsonType: 'objectId'
                },
                nombre: {
                  bsonType: 'string'
                },
                descripcion: {
                  bsonType: 'string'
                },
                costo: {
                  bsonType: 'int'
                }
              }
            }
          }
        }
      }
    }
  }
```

Escenarios de prueba

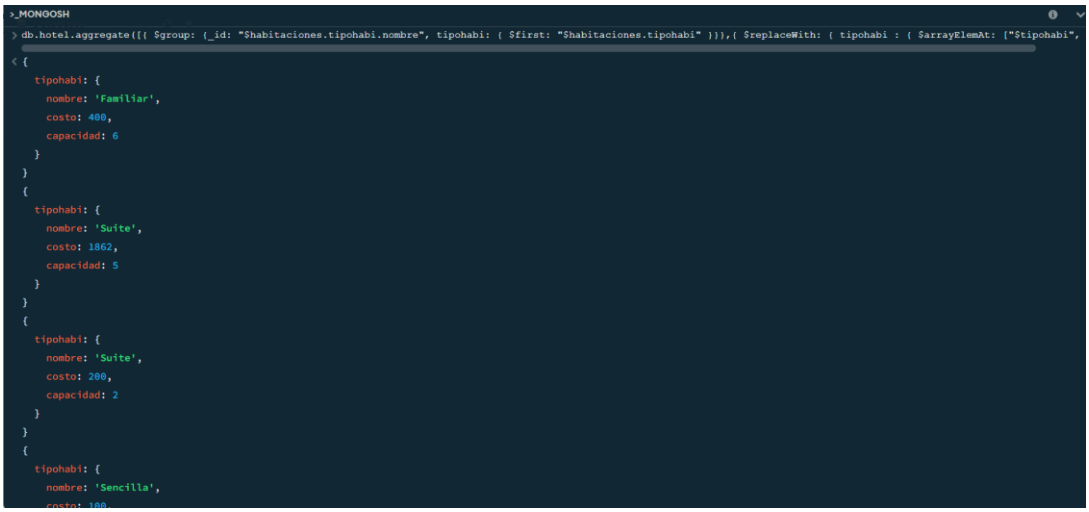
Para la inserción de información a la base de datos se uso el comando de insertMany() con la información de los documentos correspondientes para ser añadidos a la colección “hotel”. Este se puede encontrar en la sección de documentos.

A continuación, se muestra las pruebas de las consultas en MongoDB que testean el funcionamiento de las consultas:

Obtener las reservas:

```
> db.hotel.find({})
< {
    _id: ObjectId("656520bcac0904ac3054e044"),
    inicio: 2023-11-27T00:00:00.000Z,
    fin: 2023-11-30T00:00:00.000Z,
    diasreserva: 3,
    checkin: true,
    checkout: false,
    habitaciones: [
      {
        numero: 101,
        dotacion: [
          'TV',
          'Aire acondicionado'
        ],
        tipohabi: {
          nombre: 'Suite',
          costo: 200,
          capacidad: 2
        }
      },
      {
        numero: 102,
        dotacion: [
          'Wi-Fi',
          'Cama doble'
```

Obtener los tipos de habitaciones:



Obtener las habitaciones:

```
db.hotel.aggregate([{$group: {_id: "$habitaciones.numero", habitaciones: { $first: "$habitaciones" }}},
        {$replaceWith: { habitaciones: { $arrayElemAt: ["$habitaciones", 0] } }}])
{
  habitaciones: {
    numero: 167,
    dotacion: [
      'Aire acondicionado'
    ],
    tipohabi: {
      nombre: 'Suite',
      costo: 856,
      capacidad: 5
    }
  }
}
{
  habitaciones: {
    numero: 112,
    dotacion: [
      'TV',
      'Aire acondicionado'
    ],
    tipohabi: {
      nombre: 'Sencilla',
      costo: 1060,
```

Obtener la información de los servicios:

```
db.hotel.aggregate([{$unwind: "$consumos"},
        {$unwind: "$consumos.servicio"},
        {$group: { _id: "$consumos.servicio.nombre", descripcion: { $first: "$consumos.servicio.descripcion" }
    ])
{
  _id: 'Restaurante',
  descripcion: 'Almuerzo',
  costo: 50,
  totalConsumos: 30
}
{
  _id: 'Prestamo de Utensilios',
  descripcion: 'BBQ',
  costo: 31,
  totalConsumos: 28
}
{
  _id: 'Tiendas',
  descripcion: 'Ropa',
  costo: 290,
  totalConsumos: 35
}
{
  _id: 'Salones',
  descripcion: 'Reunion',
```

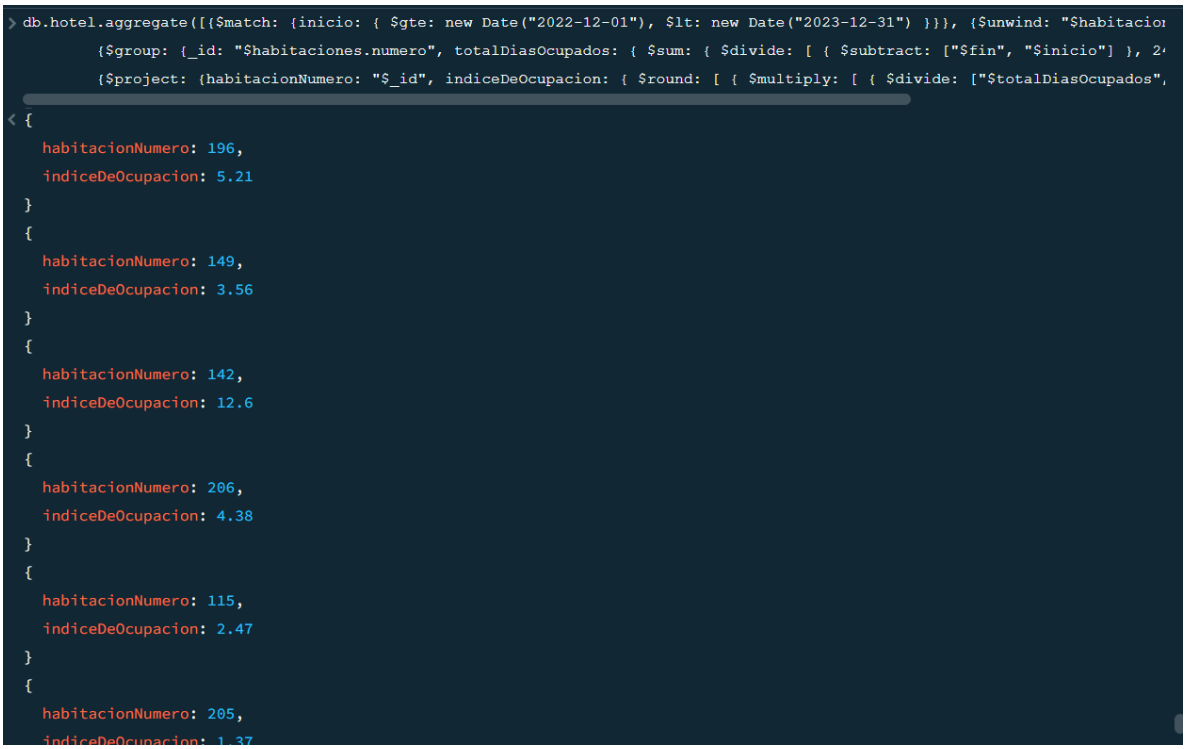
Obtener consumos:

```
> db.hotel.aggregate([{ $unwind: {path: "$consumos" }}])
< {
    _id: ObjectId("656520bcac0904ac3054e044"),
    inicio: 2023-11-27T00:00:00.000Z,
    fin: 2023-11-30T00:00:00.000Z,
    diasreserva: 3,
    checkin: true,
    checkout: false,
    habitaciones: [
      {
        numero: 101,
        dotacion: [
          'TV',
          'Aire acondicionado'
        ],
        tipohabi: {
          nombre: 'Suite',
          costo: 200,
          capacidad: 2
        }
      },
      {
        numero: 102,
        dotacion: [
          'Wi-Fi',
          'Cama doble'
```

Obtener RFC1

```
> db.hotel.aggregate([{$unwind: "$consumos"},
        {$group: {_id: "$habitaciones.numero", totalDineroRecolectado: { $sum: "$consumos.servicio.costo" }}}])
< {
    _id: [
      190
    ],
    totalDineroRecolectado: 103
  }
  {
    _id: [
      146
    ],
    totalDineroRecolectado: 774
  }
  {
    _id: [
      148
    ],
    totalDineroRecolectado: 104
  }
  {
    _id: [
      142
    ],
    totalDineroRecolectado: 1682
  }
```

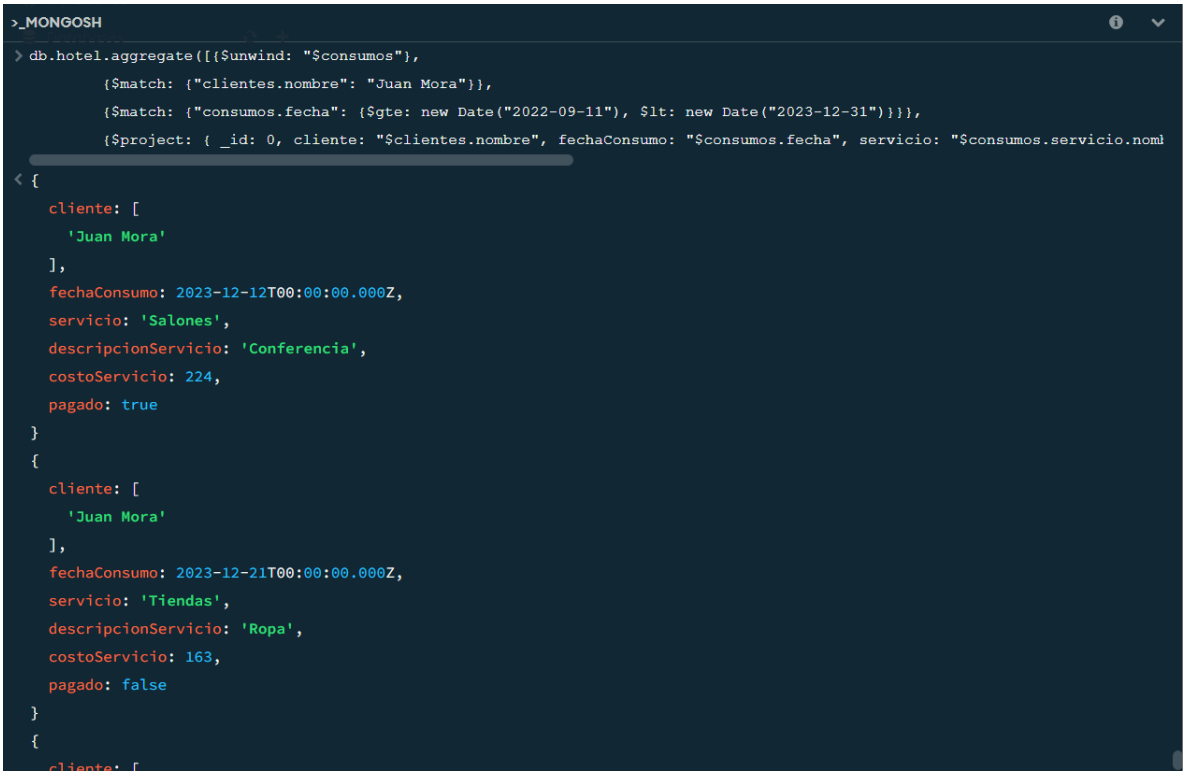
Obtener RFC2



```
> db.hotel.aggregate([{$match: {inicio: { $gte: new Date("2022-12-01"), $lt: new Date("2023-12-31") }}}, {$unwind: "$habitacion
        {$group: {_id: "$habitaciones.numero", totalDiasOcupados: { $sum: { $divide: [ { $subtract: ["$fin", "$inicio"] }, 2
        {$project: {habitacionNumero: "$_id", indiceDeOcupacion: { $round: [ { $multiply: [ { $divide: ["$totalDiasOcupados",

< {
    habitacionNumero: 196,
    indiceDeOcupacion: 5.21
  }
  {
    habitacionNumero: 149,
    indiceDeOcupacion: 3.56
  }
  {
    habitacionNumero: 142,
    indiceDeOcupacion: 12.6
  }
  {
    habitacionNumero: 206,
    indiceDeOcupacion: 4.38
  }
  {
    habitacionNumero: 115,
    indiceDeOcupacion: 2.47
  }
  {
    habitacionNumero: 205,
    indiceDeOcupacion: 1.37
```

Obtener RFC3



```
>_MONGOSH
> db.hotel.aggregate([{$unwind: "$consumos"},
        {$match: {"clientes.nombre": "Juan Mora"}},
        {$match: {"consumos.fecha": {$gte: new Date("2022-09-11"), $lt: new Date("2023-12-31")}}},
        {$project: { _id: 0, cliente: "$clientes.nombre", fechaConsumo: "$consumos.fecha", servicio: "$consumos.servicio.nom

< {
    cliente: [
      'Juan Mora'
    ],
    fechaConsumo: 2023-12-12T00:00:00.000Z,
    servicio: 'Salones',
    descripcionServicio: 'Conferencia',
    costoServicio: 224,
    pagado: true
  }
  {
    cliente: [
      'Juan Mora'
    ],
    fechaConsumo: 2023-12-21T00:00:00.000Z,
    servicio: 'Tiendas',
    descripcionServicio: 'Ropa',
    costoServicio: 163,
    pagado: false
  }
  {
    cliente: [
```

Test de inserción

Para cada una de las entidades se testeo que se cumpliera con los requisitos descritos en el validador. Esto implica que los tipos de datos de cada atributo sean los correctos y que existan por lo menos aquellos atributos requeridos para que la aplicación funcione correctamente. A continuación, se presenta el resultado de insertar una reserva con el costo de uno de los servicios que se consumieron con el tipo de dato incorrecto, se observó un string cuando se esperaba un int. Como este error es el que sale para todos los casos en los que no se cumple con la validación solo se mostrará un pantallazo pero se verificó su funcionalidad para todas las entidades.