Entrega 3 Proyecto – Hotel de los Andes

Documento de informe y diseño

- Julián David Bohórquez Beltrán - 202121973
- Santiago Rodríguez Mora - 202110332
- Valeria Torres Gómez – 202110363

Contenido

Diseño y construcción de la aplicación

Elementos fundamentales parte del negocio descrito

Tipos de Habitaciones: Se mencionan diferentes tipos de habitaciones, como suite presidencial, suite, familiar, doble y sencilla, cada una con características y capacidades específicas.

Servicios Adicionales del Hotel: Se mencionan diferentes servicios adicionales, como piscina, gimnasio, internet, bares, restaurantes, supermercado, tiendas, SPA, servicios de lavado/planchado/embolada, préstamo de utensilios, salones de reuniones y salones de conferencias.

Operaciones Principales: Las operaciones principales del hotel incluyen check-in, check-out, reserva de habitaciones, entrada de clientes, consumo de productos y servicios, salida de clientes y reserva de servicios adicionales.

Consumo de Productos y Servicios: En el caso de estudio se destaca el consumo de productos y servicios que se carga a la cuenta de la habitación del cliente, como alimentos, bebidas y servicios de SPA.

Reservas de Alojamiento: Se menciona que la aplicación debe permitir la gestión de reservas de habitaciones por parte de los clientes, especificando fechas de entrada y salida, así como el número de personas.

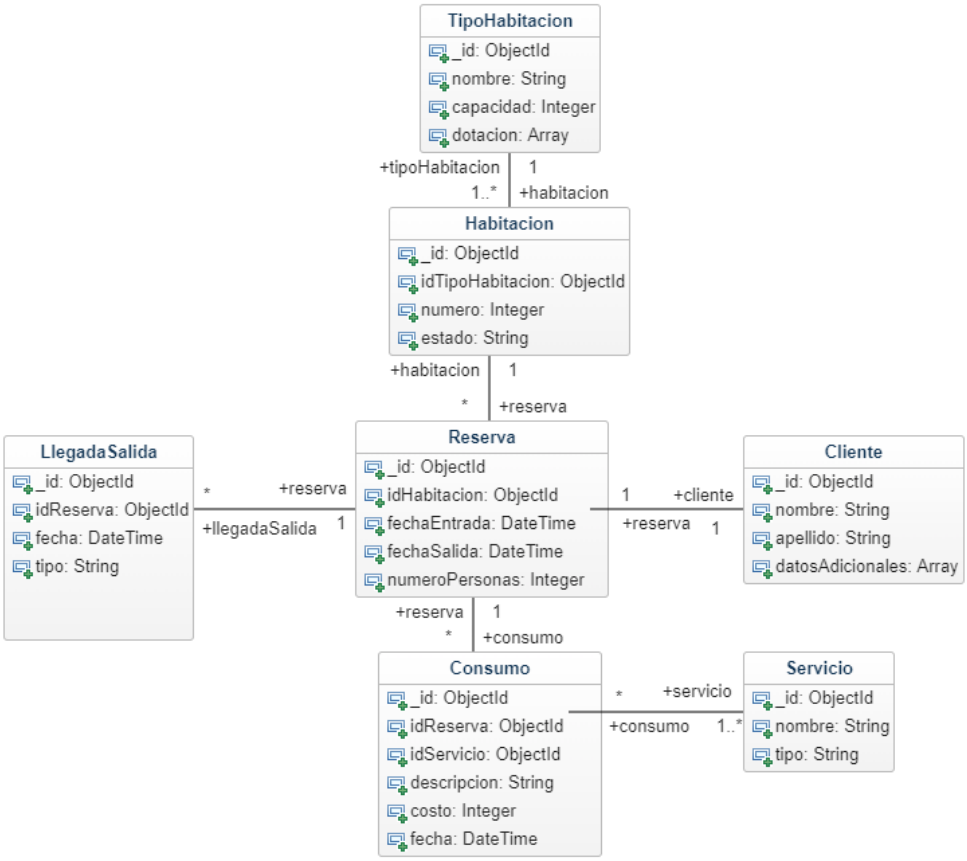
Requerimientos Funcionales (RF): Se destacan los requisitos funcionales de la aplicación, que incluyen operaciones CRUD para tipos de habitaciones, habitaciones, servicios del hotel, reservas de alojamiento, llegada de clientes, consumo de servicios y salida de clientes.

Requerimientos Funcionales de Consulta (RFC): Se especifican requerimientos para consultas básicas y avanzadas, como estadísticas de dinero recolectado por servicios, índice de ocupación de habitaciones, consumo por cliente en un rango de fechas y consultas específicas sobre el consumo de servicios.

Requerimientos No Funcionales (RNF): Se incluyen aspectos como la persistencia de la información, la centralización de la base de datos en una sola máquina y estimaciones de carga de trabajo para diferentes operaciones.

Análisis y modelo conceptual

Modelo UML propuesto para el caso de estudio:



Diseño de la Base de datos

Análisis de la carga de trabajo

TipoHabitacion:
Atributos: _id, nombre, capacidad, dotacion.
Se estima alrededor de 20 registros (tipos de habitación)

Habitacion:
Atributos: _id, idTipoHabitacion, numero, estado.
Se estima alrededor de 200 registros (habitaciones)

Servicio:
Atributos: _id, nombre, tipo.
Se estima alrededor de 35 registros (servicios)

Reserva:
Atributos: _id, idHabitacion, fechaEntrada, fechaSalida, numeroPersonas.
Se estima hasta 50,000 registros en un período de 3 años. Suponiendo que un hotel grande podría tener hasta 17,000 reservas por año

Cliente:
Atributos: _id, nombre, apellido, datosAdicionales.
La cantidad de registros dependerá de la cantidad de clientes. Se podría estimar alrededor de 20,000 registros para tener un margen amplio

Consumo:
Atributos: _id, idReserva, idServicio, descripcion, costo, fecha.
Se estima hasta 250,000 registros en un período de 3 años. Suponiendo que podría haber hasta 500 consumos por día

LlegadaSalida:
Atributos: _id, idReserva, fecha, tipo.
Se estima hasta 50,000 registros en un período de 3 años, basándose en la cantidad de reservas y la asociación de llegadas/salidas

Análisis de las operaciones de lectura y escritura para cada entidad:

Entities	Operations	Information Needed	Type	Rate
TipoHabitacion	Leer información de un tipo de habitacion	Detalles tipo de habitacion	Read	1/sec
TipoHabitacion	Actualizar detalles de un tipo de habitacion	Nueva información tipo de habitacion	Write	0.1/sec

Habitacion	Leer información de una habitación	Detalles de la habitación	Read	10/sec
Habitacion	Actualizar detalles de una habitación	Nueva información de habitación	Write	1/sec
Servicio	Leer información de un servicio	Detalles del servicio	Read	5/sec
Servicio	Actualizar detalles de un servicio	Nueva información del servicio	Write	0.5/sec
Reserva	Leer detalles de una reserva	Detalles de la reserva	Read	20/sec
Reserva	Realizar una nueva reserva	Datos de la nueva reserva	Write	2/sec
Reserva	Actualizar detalles de una reserva	Nueva información de la reserva	Write	1/sec
Reserva	Consultar reservas en un rango de fechas	Lista de reservas fechas	Read	5/sec
Reserva	Consultar reservas de un cliente en especifico	Lista de reservas cliente	Read	3/sec
Cliente	Leer detalles de un cliente	Detalles del cliente	Read	15/sec
Cliente	Actualizar detalles de un cliente	Nueva información del cliente	Write	1/sec
Consumo	Leer detalles de un consumo	Detalles del consumo	Read	30/sec
Consumo	Registrar un nuevo consumo	Datos del nuevo consumo	Write	5/sec
Consumo	Consultar consumos de un cliente en especifico	Lista de consumos cliente	Read	8/sec
LlegadaSalida	Leer detalles de llegadas/salidas	Detalles de Llegadas/salidas	Read	25/sec
LlegadaSalida	Registrar llegada/salida	Datos de la nueva llegada/salida	Write	3/sec
LlegadaSalida	Consultar llegadas\salidas en un rango de fechas	Lista de llegadas/salidas fechas	Read	12/sec

Leer información de un tipo de habitación: Se estima que esta operación podría ocurrir con una frecuencia (una vez por segundo), ya que los tipos de habitaciones no se consultan con frecuencia.

Actualizar detalles de un tipo de habitación: La actualización de los detalles de un tipo de habitación sería una operación menos frecuente en comparación con la lectura, por lo que se asignó una tasa de 0.1 operaciones por segundo.

Leer información de una habitación: La consulta de información de una habitación puede ser más frecuente, por lo que tiene una tasa más alta alrededor de 10 operaciones por segundo.

Actualizar detalles de una habitación: Similar a la actualización de tipos de habitación, la actualización de detalles de una habitación sería menos frecuente que la lectura por lo que puede tener una tasa de 1 operación por segundo.

Leer información de un servicio: La consulta de información de un servicio puede ser moderadamente frecuente con una tasa de 5 operaciones por segundo.

Actualizar detalles de un servicio: Similar a las actualizaciones anteriores, la actualización de detalles de un servicio sería menos frecuente por lo que podría tener una tasa de 0.5 operaciones por segundo.

Leer detalles de una reserva: la consulta de detalles de una reserva puede ser bastante frecuente, con una tasa de 20 operaciones por segundo.

Realizar una nueva reserva: la creación de una nueva reserva sería menos frecuente que la lectura de reservas existentes, quizá con una tasa de 2 operaciones por segundo.

Actualizar detalles de una reserva: Similar a las actualizaciones anteriores la actualización de detalles de una reserva sería menos frecuente, por lo que podría tener una tasa de 1 operación por segundo.

Consultar reservas en un rango de fechas: Supuse que esta consulta sería moderadamente frecuente, asignando una tasa de 5 operaciones por segundo.

Consultar reservas de un cliente específico: Similar a la consulta anterior, la consulta de reservas de un cliente específico sería moderadamente frecuente, a una tasa de 3 operaciones por segundo.

Leer detalles de un cliente: La lectura de detalles de un cliente sería frecuente, con una tasa de 15 operaciones por segundo.

Actualizar detalles de un cliente: Similar a las actualizaciones anteriores, la actualización de detalles de un cliente sería menos frecuente, con una tasa de 1 operación por segundo.

Leer detalles de un consumo: la lectura de detalles de un consumo sería muy frecuente en el contexto del hotel, a una tasa de 30 operaciones por segundo.

Registrar un nuevo consumo: Estimé que la creación de un nuevo consumo sería frecuente, a una tasa de 5 operaciones por segundo.

Consultar consumos de un cliente específico: Como en la consulta anterior, la consulta de consumos de un cliente específico sería moderadamente frecuente con una tasa de 8 operaciones por segundo.

Leer detalles de llegadas/salidas: La lectura de detalles de llegadas/salidas podría ser frecuente, con una tasa de 25 operaciones por segundo.

Registrar nueva llegada/salida: la creación de nuevas llegadas/salidas sería frecuente, a una tasa de 3 operaciones por segundo.

Consultar llegadas/salidas en un rango de fechas: esta consulta sería moderadamente frecuente, con una tasa de 12 operaciones por segundo.

Descripción de las entidades de datos y relaciones correspondientes al UML

TipoHabitacion:
Representa los diferentes tipos de habitaciones que el hotel ofrece, como suite presidencial, suite, familiar, doble, sencilla, etc.
Atributos:

- TipoHabitacionID (identificador único)
- Nombre
- Capacidad
- Dotacion

Habitacion:
Representa las instancias específicas de habitaciones en el hotel.
Atributos:

- HabitacionID (identificador único)
- idTipoHabitacion (clave foránea para vincular con TipoHabitacion)
- Numero
- Estado (ocupada, disponible, etc.)

Servicio:
Representa los servicios que el hotel ofrece a los clientes.
Atributos:

- ServicioID (identificador único)
- Nombre
- tipo

Reserva:
Representa las reservas realizadas por los clientes para alojarse en una habitación del hotel.
Atributos:

- ReservaID (identificador único)
- HabitacionID (clave foránea para vincular con Habitacion)
- FechaEntrada
- FechaSalida
- numeroPersonas

Cliente:
Representa a los clientes que hacen reservas y consumen servicios en el hotel.
Atributos:

- ClienteID (identificador único)
- Nombre
- apellido
- DatosAdicionales

Consumo:
Representa los consumos de servicios realizados por los clientes durante su estancia en el hotel.
Atributos:

- ConsumoID (identificador único)
- ReservaID (clave foránea para vincular con Reserva)
- ServicioID (clave foránea para vincular con Servicio)
- Descripcion
- Costo
- Fecha

LlegadaSalida:
Representa las llegadas y salidas de clientes al y del hotel.
Atributos:

- LlegadaSalidaID (identificador único)
- ReservaID (clave foránea para vincular con Reserva)
- Tipo (llegada o salida)
- Fecha

Relaciones:

TipoHabitacion - Habitacion:
Uno a muchos, ya que un tipo de habitación puede tener varias habitaciones asociadas.

Habitacion - Reserva:

Uno a muchos, ya que una habitación puede estar asociada a varias reservas, pero una reserva está asociada a una habitación.

Reserva - Consumo:
Uno a muchos, ya que una reserva puede tener múltiples consumos, pero un consumo está asociado a una reserva.

Reserva - Cliente:
Uno a uno, ya que un cliente realiza una reserva, y una reserva está asociada a un cliente.

Consumo - Servicio:
Uno a uno o muchos a uno, teniendo en cuenta que se pueden tener varios servicios.

Reserva - LlegadaSalida:
Uno a muchos, ya que una reserva puede tener múltiples registros de llegada o salida, pero un registro está asociado a una reserva.

Análisis de selección de esquema de asociación

TipoHabitacion - Habitacion

Simplicity (Embed: Sí, Reference: No): Mantener la información de tipos de habitaciones y habitaciones juntas conduce a un modelo más simple, ya que las habitaciones son inherentemente parte de un tipo específico de habitación. No hay necesidad de buscar o unir datos separados.

Go Together (Embed: Sí, Reference: No): Las habitaciones tienen una relación "tiene un" con los tipos de habitaciones. Cada habitación pertenece a un tipo específico de habitación, estableciendo una relación directa.

Query Atomicity (Embed: Sí, Reference: No): Las consultas sobre la disponibilidad de habitaciones generalmente incluirían información sobre los tipos de habitaciones. Consultar ambas entidades juntas es más eficiente.

Update Complexity (Embed: Sí, Reference: No): Actualizar la información de tipos de habitaciones generalmente implicaría actualizar las habitaciones asociadas. Mantenerlas juntas simplifica el proceso de actualización.

Archival (Embed: Sí, Reference: No): Al archivar información histórica, tiene sentido archivar tanto los tipos de habitaciones como las habitaciones asociadas al mismo tiempo.

Cardinality (Embed: No, Reference: Sí): La cardinalidad en el lado de las habitaciones no es alta, ya que cada habitación está asociada con un tipo específico. Sin embargo, en el lado de los tipos de habitaciones, la cardinalidad podría ser más alta.

Data Duplication (Embed: No, Reference: Sí): La duplicación de datos no sería complicada en este caso, ya que cada tipo de habitación se gestiona de forma independiente.

Document Size (Embed: Moderado, Reference: Sí): El tamaño del documento no sería excesivo si las habitaciones se incrustaran bajo los tipos de habitaciones. Sin embargo, si se manejan por separado, podría haber un tamaño considerable de documentos referenciados.

Document Growth (Embed: No, Reference: Sí): Las habitaciones no crecen sin límites bajo un tipo de habitación específico. La referencia es más adecuada si se espera un crecimiento considerable.

Workload (Embed: Sí, Reference: No): En un entorno de carga de escritura pesada, actualizar la disponibilidad de las habitaciones podría ser más frecuente. Mantenerlas incrustadas facilita estas actualizaciones.

Individuality (Embed: No, Reference: Sí): Las habitaciones generalmente no existen de manera significativa sin un tipo de habitación específico. Referenciar proporciona independencia.

Habitación - Reserva

Simplicity (Embed: No, Reference: Sí): Mantener la información de habitaciones y reservas por separado lleva a un modelo más simple, ya que las habitaciones y las reservas son entidades independientes.

Go Together (Embed: No, Reference: Sí): Las habitaciones y las reservas no tienen una relación "tiene un" o "contiene" directa. Cada entidad existe independientemente.

Query Atomicity (Embed: No, Reference: Sí): Las consultas sobre habitaciones y reservas generalmente se realizan por separado. La información no se consulta de manera atómica.

Update Complexity (Embed: No, Reference: Sí): Actualizar información de habitaciones y reservas no suele hacerse al mismo tiempo. Mantenerlos separados facilita las actualizaciones específicas.

Archival (Embed: No, Reference: Sí): La información histórica sobre habitaciones y reservas puede archivarse de manera independiente, ya que no están intrínsecamente relacionadas.

Cardinality (Embed: No, Reference: Sí): La cardinalidad en el lado de las reservas puede ser alta, ya que una reserva puede estar asociada a varias habitaciones. En cambio, las habitaciones pueden estar asociadas a una reserva específica.

Data Duplication (Embed: No, Reference: Sí): La duplicación de datos no sería complicada, ya que cada entidad puede gestionarse de manera independiente.

Document Size (Embed: Moderado, Reference: Sí): El tamaño del documento no sería excesivo si las habitaciones se referenciaran. Mantenerlas por separado permite un tamaño de documento más manejable.

Document Growth (Embed: No, Reference: Sí): Las habitaciones no crecen sin límites bajo una reserva específica. La referencia es más adecuada si se espera un crecimiento considerable.

Workload (Embed: No, Reference: Sí): En un entorno de carga de escritura pesada, actualizar habitaciones y reservas por separado puede ser más eficiente.

Individuality (Embed: No, Reference: Sí): Las habitaciones generalmente pueden existir independientemente de una reserva específica. Referenciar proporciona independencia.

Reserva - Consumo

Simplicity (Embed: No, Reference: Sí): Mantener la información de reservas y consumos por separado lleva a un modelo más simple, ya que son entidades independientes.

Go Together (Embed: No, Reference: Sí): Las reservas y los consumos no tienen una relación "tiene un" o "contiene" directa. Cada entidad existe independientemente.

Query Atomicity (Embed: No, Reference: Sí): Las consultas sobre reservas y consumos generalmente se realizan por separado. La información no se consulta de manera atómica.

Update Complexity (Embed: No, Reference: Sí): Actualizar información de reservas y consumos no suele hacerse al mismo tiempo. Mantenerlos separados facilita las actualizaciones específicas.

Archival (Embed: No, Reference: Sí): La información histórica sobre reservas y consumos puede archivarse de manera independiente, ya que no están intrínsecamente relacionadas.

Cardinality (Embed: No, Reference: Sí): La cardinalidad en el lado de los consumos puede ser alta, ya que un consumo puede estar asociado a una reserva específica. En cambio, las reservas pueden estar asociadas a varios consumos.

Data Duplication (Embed: No, Reference: Sí): La duplicación de datos no sería complicada, ya que cada entidad puede gestionarse de manera independiente.

Document Size (Embed: Moderado, Reference: Sí): El tamaño del documento no sería excesivo si los consumos se referenciaran. Mantenerlos por separado permite un tamaño de documento más manejable.

Document Growth (Embed: No, Reference: Sí): Los consumos no crecen sin límites bajo una reserva específica. La referencia es más adecuada si se espera un crecimiento considerable.

Workload (Embed: No, Reference: Sí): En un entorno de carga de escritura pesada, actualizar reservas y consumos por separado puede ser más eficiente.

Individuality (Embed: No, Reference: Sí): Los consumos generalmente pueden existir independientemente de una reserva específica. Referenciar proporciona independencia.

Reserva - Cliente

Simplicity (Embed: Sí, Reference: No): Mantener la información de reservas y clientes juntas conduce a un modelo más simple, ya que cada reserva está directamente asociada a un cliente.

Go Together (Embed: Sí, Reference: No): Cada reserva tiene una relación directa con un cliente específico. La relación es claramente "tiene un" o "pertenece a".

Query Atomicity (Embed: Sí, Reference: No): Las consultas sobre reservas generalmente incluirían información sobre clientes. Consultar ambas entidades juntas es más eficiente.

Update Complexity (Embed: Sí, Reference: No): Actualizar información de reservas y clientes generalmente se hace al mismo tiempo. Mantenerlos juntos facilita las actualizaciones específicas.

Archival (Embed: Sí, Reference: No): Al archivar información histórica, tiene sentido archivar tanto las reservas como los clientes asociados al mismo tiempo.

Cardinality (Embed: No, Reference: Sí): La cardinalidad en el lado de los clientes puede ser baja, ya que un cliente está asociado a una reserva específica. En cambio, las reservas pueden estar asociadas a múltiples clientes si se permite la modificación de la reserva.

Data Duplication (Embed: No, Reference: Sí): La duplicación de datos no sería complicada, ya que cada reserva está asociada a un cliente específico.

Document Size (Embed: Moderado, Reference: Sí): El tamaño del documento no sería excesivo si los clientes se referenciaran. Mantenerlos por separado permite un tamaño de documento más manejable.

Document Growth (Embed: No, Reference: Sí): Los clientes no crecen sin límites bajo una reserva específica. La referencia es más adecuada si se espera un crecimiento considerable.

Workload (Embed: Sí, Reference: No): En un entorno de carga de escritura pesada, actualizar reservas y clientes por separado puede ser más

Consumo - Servicio

Simplicity (Embed: Sí, Reference: No): Mantener la información de consumos y servicios juntas conduciría a un modelo más simple, ya que los consumos están directamente relacionados con los servicios del hotel.

Go Together (Embed: Sí, Reference: No): Los consumos de servicios tienen una relación directa con los servicios del hotel. Cada consumo está vinculado a un servicio específico.

Query Atomicity (Embed: Sí, Reference: No): Las consultas sobre consumos de servicios generalmente incluirían información sobre los servicios del hotel. Consultar ambas entidades juntas es más eficiente.

Update Complexity (Embed: Sí, Reference: No): Actualizar información de servicios del hotel y consumos de servicios por cliente o acompañantes generalmente se hace al mismo tiempo. Mantenerlos juntos facilita las actualizaciones específicas.

Archival (Embed: Sí, Reference: No): Al archivar información histórica, tiene sentido archivar tanto los servicios del hotel como los consumos asociados al mismo tiempo.

Cardinality (Embed: No, Reference: Sí): La cardinalidad en el lado de los consumos de servicios puede ser alta, ya que un cliente puede realizar múltiples consumos. En cambio, los servicios del hotel pueden ser independientes de los consumos específicos.

Data Duplication (Embed: No, Reference: Sí): La duplicación de datos no sería complicada en este caso, ya que cada servicio puede gestionarse de manera independiente.

Document Size (Embed: Moderado, Reference: Sí): El tamaño del documento no sería excesivo si los consumos de servicios se incrustaran bajo los servicios del hotel. Sin embargo, si se manejan por separado, podría haber un tamaño considerable de documentos referenciados.

Document Growth (Embed: No, Reference: Sí): Los consumos de servicios no crecen sin límites bajo un servicio específico. La referencia es más adecuada si se espera un crecimiento considerable.

Workload (Embed: Sí, Reference: No): En un entorno de carga de escritura pesada, actualizar consumos de servicios y servicios por separado puede ser más eficiente.

Individuality (Embed: No, Reference: Sí): Los consumos de servicios generalmente no existen de manera significativa sin un servicio específico. Referenciar proporciona independencia.

Relación: Reserva - LlegadaSalida

Simplicity (Embed: No, Reference: Sí): Mantener la información de reservas y registros de llegada o salida por separado conduce a un modelo más simple, ya que son entidades independientes.

Go Together (Embed: No, Reference: Sí): Los registros de llegada o salida no tienen una relación intrínseca con las reservas. Cada registro está vinculado a una reserva, pero no es necesario que estén incrustados.

Query Atomicity (Embed: No, Reference: Sí): Las consultas sobre registros de llegada o salida no siempre incluirán información sobre la reserva. Consultar ambas entidades por separado puede ser más eficiente.

Update Complexity (Embed: No, Reference: Sí): Actualizar información de registros de llegada o salida y reservas generalmente se hace en momentos diferentes. Mantenerlos por separado facilita las actualizaciones específicas.

Archival (Embed: No, Reference: Sí): Al archivar información histórica, tiene sentido archivar las reservas y los registros de llegada o salida por separado.

Cardinality (Embed: No, Reference: Sí): La cardinalidad en el lado de los registros de llegada o salida puede ser alta, ya que una reserva puede tener múltiples registros. En cambio, las reservas son únicas.

Data Duplication (Embed: No, Reference: Sí): La duplicación de datos no sería complicada en este caso, ya que cada registro puede gestionarse de manera independiente.

Document Size (Embed: No, Reference: Sí): El tamaño del documento no sería excesivo si los registros de llegada o salida se manejaran por separado de las reservas.

Document Growth (Embed: No, Reference: Sí): Los registros de llegada o salida pueden crecer sin límites. La referencia es más adecuada si se espera un crecimiento considerable.

Workload (Embed: No, Reference: Sí): En un entorno de carga de escritura pesada, actualizar registros de llegada o salida y reservas por separado puede ser más eficiente.

Individuality (Embed: No, Reference: Sí): Los registros de llegada o salida generalmente no existen de manera significativa sin una reserva específica. Referenciar proporciona independencia.

Descripción grafica JSON

Esquema de Asociación (Embed)


```
{
  "TipoHabitacion": {
    "habitaciones": [
      {
        "HabitacionID": 101,
        "Numero": 101,
        "Estado": "disponible"
      },
      {
        "HabitacionID": 102,
        "Numero": 102,
        "Estado": "ocupada"
      }
    ]
  }
}
```


Esquema de Asociación (Embed)


```
{
  "Habitacion": {
    "reservas": [
      {
        "ReservaID": "R001",
        "FechaEntrada": "2023-04-01",
        "FechaSalida": "2023-04-05",
        "NumeroPersonas": 2
      },
      {
        "ReservaID": "R002",
        "FechaEntrada": "2023-04-03",
        "FechaSalida": "2023-04-08",
        "NumeroPersonas": 1
      }
    ]
  }
}
```

Esquema de Asociación (Embed)


```
{
  "Reserva": {
    "consumos": [
      {
        "ConsumoID": 1,
        "ServicioID": 101,
        "Descripcion": "Cóctel Margarita",
        "Costo": 15.99,
        "Fecha": "2023-04-02"
      },
      {
        "ConsumoID": 2,
        "ServicioID": 102,
        "Descripcion": "Filete Mignon",
        "Costo": 29.99,
        "Fecha": "2023-04-05"
      }
    ]
  }
}
```


Esquema de Asociación (Embed)


```
{
  "Reserva": {
    "consumos": [
      {
        "ConsumoID": 1,
        "ServicioID": 101,
        "Descripcion": "Cóctel Margarita",
        "Costo": 15.99,
        "Fecha": "2023-04-02"
      },
      {
        "ConsumoID": 2,
        "ServicioID": 102,
        "Descripcion": "Filete Mignon",
        "Costo": 29.99,
        "Fecha": "2023-04-05"
      }
    ]
  }
}
```

Esquema de Asociación (Embed)

```
{
  "Consumo": {
    "servicio": {
      "ServicioID": 201,
      "Nombre": "SPA",
      "Tipo": "Masaje"
    },
    "Descripcion": "Sesión de masaje de 60 minutos",
    "Costo": 50.00,
    "Fecha": "2023-04-12"
  }
}
```

Esquema de Asociación (Reference)

```
{
  "Reserva": {
    "llegadasSalidas": [
      {
        "LlegadaSalidaID": 1,
        "Tipo": "Llegada",
        "Fecha": "2023-04-10T15:00:00"
      },
      {
        "LlegadaSalidaID": 2,
        "Tipo": "Salida",
        "Fecha": "2023-04-15T11:00:00"
      }
    ]
  }
}
```

Creación de las colecciones principales

```
db.createCollection("tipoHabitacion", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nombre", "capacidad", "dotacion"],
      properties: {
        nombre: { bsonType: "string" },
        capacidad: { bsonType: "int" },
        dotacion: { bsonType: "string" },
      },
```

```
    },
  },
});

db.createCollection("habitacion", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["idTipoHabitacion", "numero", "estado"],
      properties: {
        idTipoHabitacion: { bsonType: "objectId" },
        numero: { bsonType: "int" },
        estado: { enum: ["ocupada", "disponible", "mantenimiento"] },
      },
    },
  },
});

db.createCollection("servicio", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nombre", "tipo"],
      properties: {
        nombre: { bsonType: "string" },
        tipo: { bsonType: "string" },
      },
    },
  },
});

db.createCollection("reserva", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["idHabitacion", "fechaEntrada", "fechaSalida", "numeroPersonas"],
      properties: {
        idHabitacion: { bsonType: "objectId" },
        fechaEntrada: { bsonType: "date" },
        fechaSalida: { bsonType: "date" },
        numeroPersonas: { bsonType: "int" },
      },
    },
  },
```

```
});

db.createCollection("cliente", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nombre", "apellido", "datosAdicionales"],
      properties: {
        nombre: { bsonType: "string" },
        apellido: { bsonType: "string" },
        datosAdicionales: { bsonType: "string" },
      },
    },
  },
});


db.createCollection("consumo", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["idReserva", "idServicio", "descripcion", "costo", "fecha"],
      properties: {
        idReserva: { bsonType: "objectId" },
        idServicio: { bsonType: "objectId" },
        descripcion: { bsonType: "string" },
        costo: { bsonType: "double" },
        fecha: { bsonType: "date" },
      },
    },
  },
});

db.createCollection("llegadaSalida", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["idReserva", "fecha", "tipo"],
      properties: {
        idReserva: { bsonType: "objectId" },
        fecha: { bsonType: "date" },
        tipo: { enum: ["llegada", "salida"] },
      },
    },
```

```
  },
});
```

Implementación de los requerimientos funcionales de CRUD y consultas

RF1 - Tipo de Habitación
Crear (Registrar) un nuevo tipo de habitación:

```
db.TipoHabitaciones.insert({
  nombre: "Suite"
});
```

Leer (Consultar) los tipos de habitaciones:
```
db.TipoHabitaciones.find({});
```

Actualizar un tipo de habitación:
```
db.TipoHabitaciones.update(
  { _id: ObjectId("id_del_tipo_de_habitacion") },
  { $set: { nombre: "Deluxe Suite" } }
);
```

Eliminar un tipo de habitación:
```
db.TipoHabitaciones.remove({ _id: ObjectId("id_del_tipo_de_habitacion") });
```

RF2 - Habitación
Crear (Registrar) una nueva habitación:
```
db.Habitaciones.insert({
  numero: 101,
  tipoHabitacion: ObjectId("id_del_tipo_de_habitacion"),
});
```

Leer (Consultar) las habitaciones:
```
db.Habitaciones.find({});
```

Actualizar una habitación:
```
db.Habitaciones.update(
  { _id: ObjectId("id_de_la_habitacion") },
  { $set: { numero: 102 } }
);
```

Eliminar una habitación:

```
db.Habitaciones.remove({ _id: ObjectId("id_de_la_habitacion") });
```

RF3 - Servicio del Hotel

Crear (Registrar) un nuevo servicio:

```
db.Servicios.insert({
  nombre: "Spa",
});
```

Leer (Consultar) los servicios:

```
db.Servicios.find({});
```

Actualizar un servicio:

```
db.Servicios.update(
  { _id: ObjectId("id_del_servicio") },
  { $set: { nombre: "Bar" } }
);
```

Eliminar un servicio:

```
db.Servicios.remove({ _id: ObjectId("id_del_servicio") });
```

RF4 - Reserva de Alojamiento

Crear (Registrar) una nueva reserva:

```
db.Reservas.insert({
  usuario: ObjectId("id_del_usuario"),
  habitacion: ObjectId("id_de_la_habitacion"),
  fecha_inicio: new Date("2023-12-20"),
  fecha_fin: new Date("2023-12-25"),
});
```

Leer (Consultar) las reservas:

```
db.Reservas.find({});
```

Actualizar una reserva:

```
db.Reservas.update(
  { _id: ObjectId("id_de_la_reserva") },
  { $set: { fecha_fin: new Date("2023-12-26") } }
);
```

Eliminar una reserva:

```
db.Reservas.remove({ _id: ObjectId("id_de_la_reserva") });
```

RF5 - Llegada de un Cliente al Hotel

Registrar llegada:

```
db.Reservas.update(
  { _id: ObjectId("id_de_la_reserva") },
  { $set: { llegada: true } }
);
```

Consultar llegadas:

```
db.Reservas.find({ llegada: true });
```

Actualizar una llegada:

```
db.Reservas.update(
  { _id: ObjectId("id_de_la_reserva"), llegada: true },
  { $set: { "detalles.llegada": new Date("2023-12-20T14:00:00Z") } }
);
```

Eliminar registro de llegada:

javascript
Copy code

```
db.Reservas.update(
  { _id: ObjectId("id_de_la_reserva") },
  { $unset: { llegada: "" } }
);
```

RF7 - Salida de un Cliente

Registrar salida:

```
db.Reservas.update(
```

```
  { _id: ObjectId("id_de_la_reserva") },
  { $set: { salida: true } }
);
```

Consultar salidas:

```
db.Reservas.find({ salida: true });
```

Actualizar registro de salida:

```
db.Reservas.update(
  { _id: ObjectId("id_de_la_reserva"), salida: true },
  { $set: { "detalles.salida": new Date("2023-12-25T10:00:00Z") } }
);
```

Eliminar registro de salida:

```
db.Reservas.update(
  { _id: ObjectId("id_de_la_reserva") },
  { $unset: { salida: "" } }
);
```

REQUERIMIENTOS FUNCIONALES DE CONSULTA BÁSICOS

RFC1 - Dinero Recogido por Servicios en Cada Habitación en el Último Año######

```
db.Reservas.aggregate([
  {
    $match: {
      fecha_inicio: {
        $gte: new Date('2023-01-01T00:00:00.000Z')
      }
    }
  },
  {
    $unwind: "$consumos"
  },
  {
    $group: {
```

```
    _id: "$habitacion",
    dineroRecolectado: { $sum: "$consumos.valorTotal" }
   }
  }
]);
```

RFC2 - Índice de Ocupación de Cada Habitación desde el 1 de Enero de 2023

```
db.Reservas.aggregate([
 {
  $match: {
   fecha_inicio: {
    $gte: new Date('2023-01-01T00:00:00.000Z')
   }
  }
 },
 {
  $project: {
   habitacion: 1,
   diasOcupada: {
    $dateDiff: {
     startDate: {
      $max: ["$fecha_inicio", new Date('2022-01-01T00:00:00.000Z')]
     },
     endDate: {
      $min: ["$fecha_fin", new Date()]
     },
     unit: "day"
    }
   }
  }
 },
 {
  $group: {
   _id: "$habitacion",
   diasTotalesOcupados: { $sum: "$diasOcupada" }
  }
 },
 {
  $project: {
   indiceOcupacion: {
    $divide: ["$diasTotalesOcupados", {
```

```
        $dateDiff: {
          startDate: new Date('2022-01-01T00:00:00.000Z'),
          endDate: new Date(),
          unit: "day"
        }
       }]
     }
    }
  }
]);
```

RFC3 - Consumo en el Hotel por un Cliente desde el 1 de Enero de 2023

```
db.Consumos.aggregate([
  {
    $match: {
      fecha: {
        $gte: new Date('2023-01-01T00:00:00.000Z'),
        $lte: new Date('2024-01-01T00:00:00.000Z')
      },
      usuario: ObjectId("id_del_cliente")
    }
  },
  {
    $group: {
      _id: "$usuario",
      consumoTotal: { $sum: "$valorTotal" }
    }
  }
]);
```

RFC4 - CONSULTAR CONSUMO EN HOTELANDES

ESCENARIOS DE PRUEBA:

Las pruebas de las consultas funcionan correctamente desde el shell de MongoDB, sin errores en la sintaxis. Sin embargo, al no tener datos cargados la consulta no arroja resultados.