To Do:

Analisis embed/ reference Usuario y cuenta:

\*El análisis de selección de esquema de asociación (referenciado o embebido) para cada relación

entre entidades. Para ello use la tabla de análisis vista en clase, la cual se retoma en el anexo C,

junto con los resultados del análisis de la carga de trabajo (workload), descrita antes.

\*Una descripción gráfica usando Json de cada relación entre entidades en donde presente un

ejemplo de datos junto con el esquema de asociación usado (referenciado o embebido). En el anexo

D se muestra un ejemplo de lo que se requiere.

\*Cree en MongoDB las colecciones principales de su base, así como los esquemas de validación para

cada colección. Puede usar Compass o Mongo Shell (Mongosh) para realizar este proceso. Guarde lo hecho

en un archivo. Anexe a los entregables los archivos con los scripts utilizados.

\* Implemente los 6 requerimientos funcionales de CRUD (RF1 a RF6) y los 2 requerimientos funcionales de

consulta (RFC1 y RFC2) solicitados en el documento marco. Tenga en cuenta los escenarios de prueba descritos

abajo para esta implementación. Evaluación detallada:

a. 7% para cada requerimiento funcional de modificación (total RF 42%)

b. 7% para cada requerimiento funcional de consulta (total RFC 14%).

Para la implementación pueden, si lo desean, reutilizar las clases que corresponden a la lógica y la conexión a la

base de datos usadas en entregas pasadas, siempre y cuando se hagan las adaptaciones pertinentes para la

implementación de los requerimientos de esta entrega.

\*Escenarios de prueba: realice las operaciones descritas abajo (ver escenarios de prueba) para poder realizar

pruebas sobre los RF de CRUD (RF1-RF6) así como sobre los RF de Consulta (RFC1 y RFC2) del documento marco

del caso de estudio. Anexe los scripts.

Para los RF CRUD (RF1-RF6), la aplicación debe permitir crear, actualizar, borrar y presentar datos, con el fin

de probar cada uno de los RF del documento marco. No es necesario un script para esta parte.

2. Para los RF CRUD (RF1-RF6), para cada una de las entidades, trate de insertar a través de MongoDB Shell una

tupla que no cumpla con su esquema de validación. Anexe el script a la entrega.

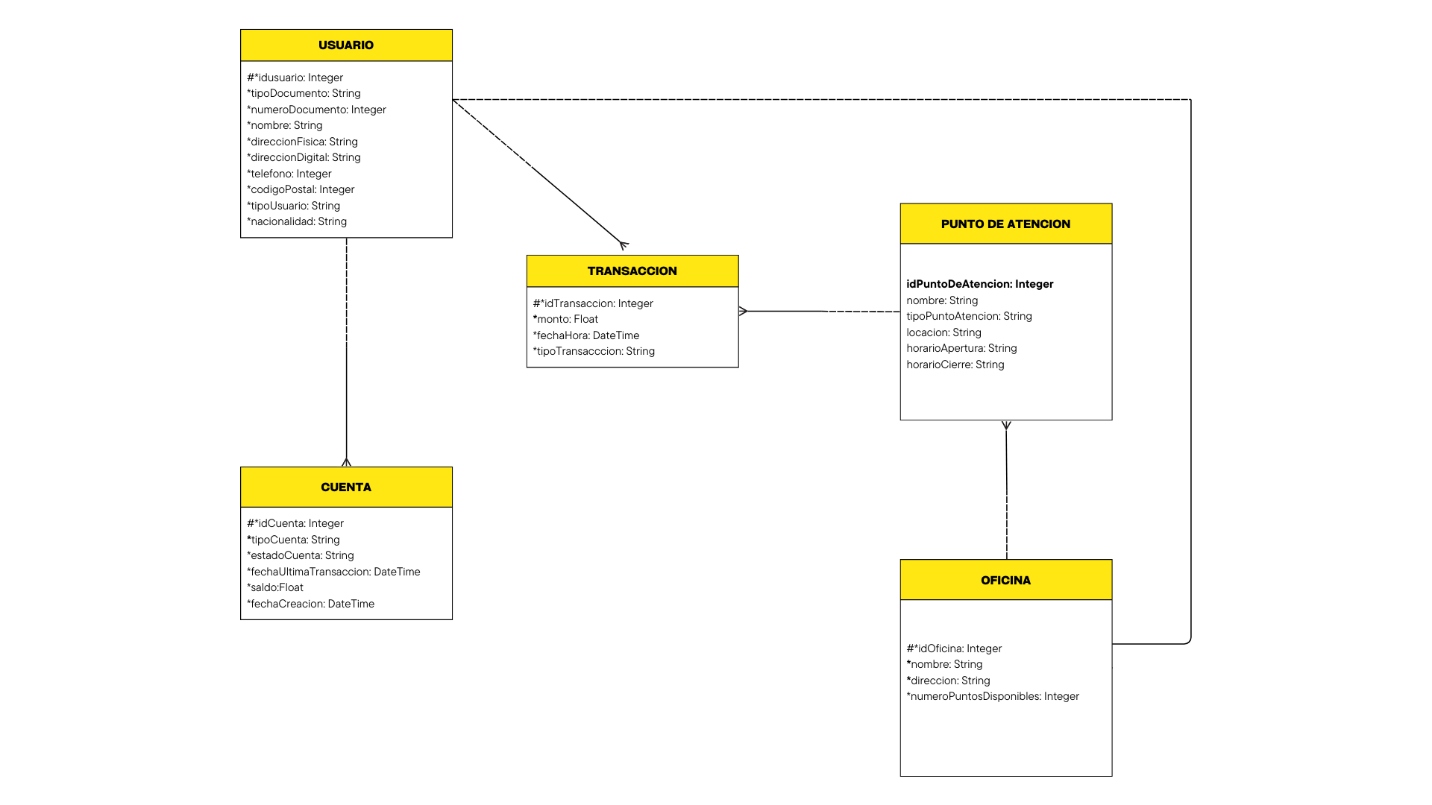
3. A través de la interfaz de la aplicación o de un script inserte suficientes datos para cada una de las entidades, de

tal manera que las consultas de los RF de Consulta (RFC1 y RFC2) puedan ser probadas. En caso de haber

usado un script, anéxelo a la entrega.

# DISEÑO Y CONSTRUCCIÓN DE LA APLICACIÓN

1. Análisis y modelo conceptual



**Imagen 1. Modelo relacional Banco de los Andes**

1. Diseño de la base de datos
   1. Análisis de la carga de trabajo.

|  |  |  |
| --- | --- | --- |
| **Entidad** | **Atributos** | **Entidades (Registros)** |
| Usuario | Id usuario, tipo documento, numero documento,  Nombre, dirección física, dirección digital, dirección, código postal, tipo usuario, nacionalidad | **1.500.000** |
| Cuenta | Id cuenta, tipo cuenta, estado cuenta, fecha ultima transacción, saldo, fecha creación | **2.500.000** |
| Transacción | Id transacción, monto, fecha hora, tipo transacción | **8.000.000** |
| Puntos de atención | Id punto de atención, nombre, tipo punto atención, dirección, horario apertura, horario cierre | **1500** |
| Oficina | Id oficina, nombre, dirección, numero puntos disponibles | **300** |

**Tabla 1. Identificación de entidades con sus atributos y cuantificación aproximada**

|  |  |  |  |
| --- | --- | --- | --- |
| **Entidad** | **Operación** | **Información necesaria** | **Tipo** |
| Usuario | Crear usuario | Atributos de usuario | Escritura |
| Oficina, Usuario | Crear Oficina | Atributos oficina + Id Usuario (gerente) | Escritura |
| Punto Atención, Oficina | Crear/Borrar | Atributos Punto Atención + Id punto de atención | Escritura |
| Cuenta, Usuario | Crear | Atributos Cuenta + Id Usuario | Escritura |
| Cuenta | Modificar Estado Cuenta | Id Cuenta | Escritura |
| Transacción, Cuenta, Usuario | Registrar operación sobre cuenta | Id Cuenta + Id Usuario+ Id Punto Atención+ Atributos transacción | Escritura |
| Cuenta | Consultar todas las cuentas | Atributos Cuentas | Lectura |
| Transacción, Cuenta | Extracto Bancario para cuenta | Id Cuenta + Saldo inicio mes + transacciones + saldo final mes | Lectura |

**Tabla 2. Operaciones de lectura y escritura para cada entidad**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entidad** | **Operación** | **Información necesaria** | **Tipo** | **Rate** |
| Usuario | Crear usuario o modificar | Atributos de usuario | Escritura | 200 veces por día |
| Usuario | Consultar usuario | Id Usuario | Lectura | 500 veces por día |
| Oficina, Usuario | Crear Oficina o modificar | Atributos oficina + Id Usuario (gerente) | Escritura | 1 vez por mes |
| Oficina, Usuario | Consultar Oficina | Id Oficina | Lectura | 1 vez por semana |
| Punto Atención, Oficina | Crear, modificar o Borrar | Atributos Punto Atención + Id punto de atención | Escritura | 1 vez por mes |
| Punto Atención | Consulta | Id punto de atención | Lectura | 1 vez por semana |
| Cuenta, Usuario | Crear | Atributos Cuenta + Id Usuario | Escritura | 500 veces por día |
| Cuenta | Modificar Estado Cuenta | Id Cuenta | Escritura | 500 veces por día |
| Transacción, Cuenta, Usuario | Registrar operación sobre cuenta | Id Cuenta + Id Usuario+ Id Punto Atención+ Atributos transacción | Escritura | 20,000 veces por día |
| Transacción, Cuenta, Usuario | Consulta operación sobre cuenta | Id Cuenta + Id usuario | Lectura | 5,000 veces por dia |
| Cuenta | Consultar cuentas | Id Cuentas | Lectura | 5,000 veces por día |
| Transacción, Cuenta | Extracto Bancario para cuenta | Id Cuenta + Saldo inicio mes + transacciones + saldo final mes | Lectura | 1 vez por mes |

**Tabla 3. Cuantificación operaciones de lectura y escritura para cada entidad**

* 1. Descripción de las entidades de datos y las relaciones NoSQL entre ellas que corresponden al modelo Er propuesto.

**Entidades:**

* Usuario: Representa la información básica de un empleado y un cliente, es decir, en el caso del empleado puede ser un gerente general, un gerente de oficina o un cajero, y en el caso del cliente puede ser natural o jurídico.
* Cuenta: Es un producto del banco que pertenece a un cliente, ya sea natural o jurídico. Las cuentas pueden ser del tipo ahorros, corriente o AFC. Además, cuentan con un estado activo, cerrado o desactivado.
* Transacción: Es realizada por un cliente en un punto de atención y afecta un producto, específicamente, a una cuenta. Los tipos de transacción son abrir, cerrar, consignar, retirar, transferir y desactivar.
* Punto De Atención: Hace referencia a las entidades por medio de las cuales se crean las entidades anteriores. Se dividen en personalizada, cajeros automáticos y digitales. Los puntos de atención personalizada de tipo cajero y los cajeros automáticos están vinculados de forma obligatoria a una oficina. Los de autoservicio digital como el portal Web y la aplicación móvil no están asociados a una oficina.
* Oficina: Punto físico que agrupa puntos de atención personalizados.

***Relaciones entre entidades y cardinalidad:***

* Usuario y cuenta: un usuario puede tener una o varias cuentas. Y, una cuenta debe tener un único usuario asociado. Por tanto, la cardinalidad es de uno a muchos.
* Usuario y transacción: Un usuario puede realizar muchas transacciones, mientras que, una transacción debe tener un usuario asociado. Por tanto, la cardinalidad es de uno a muchos.
* Transacción y punto de atención: Una transacción puede tener un punto de atención asociado, y, un punto de atención puede tener muchas transacciones. Entonces, la cardinalidad es de uno a muchos.
* Punto de atención y oficina: Un punto de atención puede o no estar asociado a una oficina. Mientras que, una oficina puede o no tener uno o varios puntos de atención asociados a ella. En conclusión, la cardinalidad es de uno a muchos.
* Oficina y usuario: La creación de una oficina tiene como requisito la afiliación de un usuario tipo gerente, el cual ya debe existir en la base de datos. Esta relación es de uno a uno.

***Análisis de selección del esquema de asociación:***

* Usuario y cuenta:

|  |  |  |
| --- | --- | --- |
| **Guideline** | **Question** | **Respuesta** |
| Simplicity | Would keeping the pieces of information together lead to a simpler data model and code? | Reference |
| Go Together | Do the pieces of information have a has-a, contains, or similar relationship? | Embed |
| Query Atomicity | Does the application query the pieces of information together? | Embed |
| Update Complexity | Are the pieces of information updated together? | Reference |
| Archival | Should the pieces of information be archived at the same time? | Reference |
| Cardinality | Is there a high cardinality in the child side of the relationship? | Reference |
| Data Duplication | Would data duplication be too complicated to manage and undesired? | Embed |
| Document size | Would the combined size of the pieces of information take too much memory or transfer bandwidth for the application? | Reference |
| Document growth | Would the embedded piece grow without bound? | Reference |
| Workload | Are the pieces of information written at different times in a Escritura-heavy workload? | Reference |
| Individuality | For the children side of the relationship, can the pieces exist by themselves without a parent? | Reference |

**Tabla 4. Reference vs Embed en la relación usuario y cuenta**

**Análisis:**  Al responder a las preguntas, se observa que, debido a las particularidades de la búsqueda, actualización y simplicidad, la mejor solución es utilizar una referencia en esta relación. Por lo tanto, la clase Usuario tendría una referencia hacia las cuentas. Sin embargo, esta solución podría generar problemas con el paso de los años. Por ello, optaremos por utilizar una referencia inversa con el fin de evitar un arreglo que crezca indefinidamente.

Es importante destacar que comprometemos la eficiencia con esta elección, ya que una referencia directa sería más eficiente. Sin embargo, debido a las características del negocio, se sacrifica la escalabilidad y es necesario poder almacenar las cuentas, es decir, se necesita garantizar la permanencia. Además, una lista infinita de cuentas podría generar conflictos debido a las restricciones de memoria que puede tener un documento en MongoDB.

En conclusión, utilizar una referencia para la relación entre usuarios y cuentas nos permitirá mantener un modelo de datos más manejable y optimizado para las consultas y actualizaciones, a pesar de los compromisos necesarios en términos de eficiencia y escalabilidad.

* Usuario y transacción:

|  |  |  |
| --- | --- | --- |
| **Guideline** | **Question** | **Respuesta** |
| Simplicity | Would keeping the pieces of information together lead to a simpler data model and code? | Reference |
| Go Together | Do the pieces of information have a has-a, contains, or similar relationship? | Embed |
| Query Atomicity | Does the application query the pieces of information together? | Embed |
| Update Complexity | Are the pieces of information updated together? | Reference |
| Archival | Should the pieces of information be archived at the same time? | Reference |
| Cardinality | Is there a high cardinality in the child side of the relationship? | Reference |
| Data Duplication | Would data duplication be too complicated to manage and undesired? | Reference |
| Document size | Would the combined size of the pieces of information take too much memory or transfer bandwidth for the application? | Reference |
| Document growth | Would the embedded piece grow without bound? | Reference |
| Workload | Are the pieces of information written at different times in a Escritura-heavy workload? | Reference |
| Individuality | For the children side of the relationship, can the pieces exist by themselves without a parent? | Reference |

**Tabla 5. Reference vs Embed en la relación usuario y transaccion**

**Análisis:** Al responder a las preguntas, se observa que, debido a las particularidades de la búsqueda, actualización y simplicidad, la mejor solución es utilizar una referencia en esta relación. Por lo tanto, la clase Transacción tendría una referencia hacia el Usuario. Sin embargo, esta solución podría generar problemas con el paso de los años. Por ello, optaremos por utilizar una referencia inversa con el fin de evitar un arreglo que crezca indefinidamente.

Es importante destacar que comprometemos la eficiencia con esta elección, ya que una referencia directa sería más eficiente. Sin embargo, debido a las características del negocio, se sacrifica la escalabilidad y es necesario poder almacenar las transacciones, es decir, se necesita garantizar la permanencia. Además, una lista infinita de transacciones podría generar conflictos debido a las restricciones de memoria que puede tener un documento en MongoDB.

En conclusión, utilizar una referencia inversa para la relación entre usuarios y transacciones nos permitirá mantener un modelo de datos más manejable y optimizado para las consultas y actualizaciones, a pesar de los compromisos necesarios en términos de eficiencia y escalabilidad

* Transacción y punto de atención:

|  |  |  |
| --- | --- | --- |
| **Guideline** | **Question** | **Respuesta** |
| Simplicity | Would keeping the pieces of information together lead to a simpler data model and code? | Reference |
| Go Together | Do the pieces of information have a has-a, contains, or similar relationship? | Embed |
| Query Atomicity | Does the application query the pieces of information together? | Embed |
| Update Complexity | Are the pieces of information updated together? | Reference |
| Archival | Should the pieces of information be archived at the same time? | Reference |
| Cardinality | Is there a high cardinality in the child side of the relationship? | Reference |
| Data Duplication | Would data duplication be too complicated to manage and undesired? | Embed |
| Document size | Would the combined size of the pieces of information take too much memory or transfer bandwidth for the application? | Reference |
| Document growth | Would the embedded piece grow without bound? | Reference |
| Workload | Are the pieces of information written at different times in a Escritura-heavy workload? | Reference |
| Individuality | For the children side of the relationship, can the pieces exist by themselves without a parent? | Reference |

**Tabla 6. Reference vs Embed en la relación transaccion y punto de atencion**

**Análisis:** Al responder a las preguntas, se observa que, debido a las particularidades de la búsqueda, actualización y simplicidad, la mejor solución es utilizar una referencia en esta relación. Por lo tanto, la clase Transacción tendría una referencia hacia el Punto de Atención. Sin embargo, esta solución podría generar problemas con el paso de los años. Por ello, optaremos por utilizar una referencia inversa con el fin de evitar un arreglo que crezca indefinidamente.

Es importante destacar que comprometemos la eficiencia con esta elección, ya que una referencia directa sería más eficiente. Sin embargo, debido a las características del negocio, se sacrifica la escalabilidad y es necesario poder almacenar las transacciones, es decir, se necesita garantizar la permanencia. Además, una lista infinita de transacciones podría generar conflictos debido a las restricciones de memoria que puede tener un documento en MongoDB.

En conclusión, utilizar una referencia inversa para la relación entre transacciones y puntos de atención nos permitirá mantener un modelo de datos más manejable y optimizado para las consultas y actualizaciones, a pesar de los compromisos necesarios en términos de eficiencia y escalabilidad

* Punto de atención y oficina:

|  |  |  |
| --- | --- | --- |
| **Guideline** | **Question** | **Respuesta** |
| Simplicity | Would keeping the pieces of information together lead to a simpler data model and code? | Reference |
| Go Together | Do the pieces of information have a has-a, contains, or similar relationship? | Embed |
| Query Atomicity | Does the application query the pieces of information together? | Embed |
| Update Complexity | Are the pieces of information updated together? | Reference |
| Archival | Should the pieces of information be archived at the same time? | Reference |
| Cardinality | Is there a high cardinality in the child side of the relationship? | Reference |
| Data Duplication | Would data duplication be too complicated to manage and undesired? | Embed |
| Document size | Would the combined size of the pieces of information take too much memory or transfer bandwidth for the application? | Reference |
| Document growth | Would the embedded piece grow without bound? | Reference |
| Workload | Are the pieces of information written at different times in a Escritura-heavy workload? | Reference |
| Individuality | For the children side of the relationship, can the pieces exist by themselves without a parent? | Reference |

**Tabla 7. Reference vs Embed en la relación oficina y punto de atención**

**Análisis:** Al responder a las preguntas, se observa que, debido a las particularidades de la búsqueda, actualización y simplicidad, la mejor solución es utilizar una referencia en esta relación. Por lo tanto, la clase Punto de Atención tendría una referencia hacia la Oficina. Sin embargo, esta solución podría generar problemas con el paso de los años. Por ello, optaremos por utilizar una referencia inversa con el fin de evitar un arreglo que crezca indefinidamente.

Es importante destacar que comprometemos la eficiencia con esta elección, ya que una referencia directa sería más eficiente. Sin embargo, debido a las características del negocio, se sacrifica la escalabilidad y es necesario poder almacenar los puntos de atención, es decir, se necesita garantizar la permanencia. Además, una lista infinita de puntos de atención podría generar conflictos debido a las restricciones de memoria que puede tener un documento en MongoDB.

En conclusión, utilizar una referencia inversa para la relación entre oficinas y puntos de atención nos permitirá mantener un modelo de datos más manejable y optimizado para las consultas y actualizaciones, a pesar de los compromisos necesarios en términos de eficiencia y escalabilidad

* Oficina y usuario:

|  |  |  |
| --- | --- | --- |
| **Guideline** | **Question** | **Respuesta** |
| Simplicity | Would keeping the pieces of information together lead to a simpler data model and code? | Reference |
| Go Together | Do the pieces of information have a has-a, contains, or similar relationship? | Embed |
| Query Atomicity | Does the application query the pieces of information together? | Embed |
| Update Complexity | Are the pieces of information updated together? | Reference |
| Archival | Should the pieces of information be archived at the same time? | Reference |
| Cardinality | Is there a high cardinality in the child side of the relationship? | Embed |
| Data Duplication | Would data duplication be too complicated to manage and undesired? | Embed |
| Document size | Would the combined size of the pieces of information take too much memory or transfer bandwidth for the application? | Reference |
| Document growth | Would the embedded piece grow without bound? | Embed |
| Workload | Are the pieces of information written at different times in a Escritura-heavy workload? | Reference |
| Individuality | For the children side of the relationship, can the pieces exist by themselves without a parent? | Reference |

**Tabla 8. Reference vs Embed en la relación oficina y usuario**

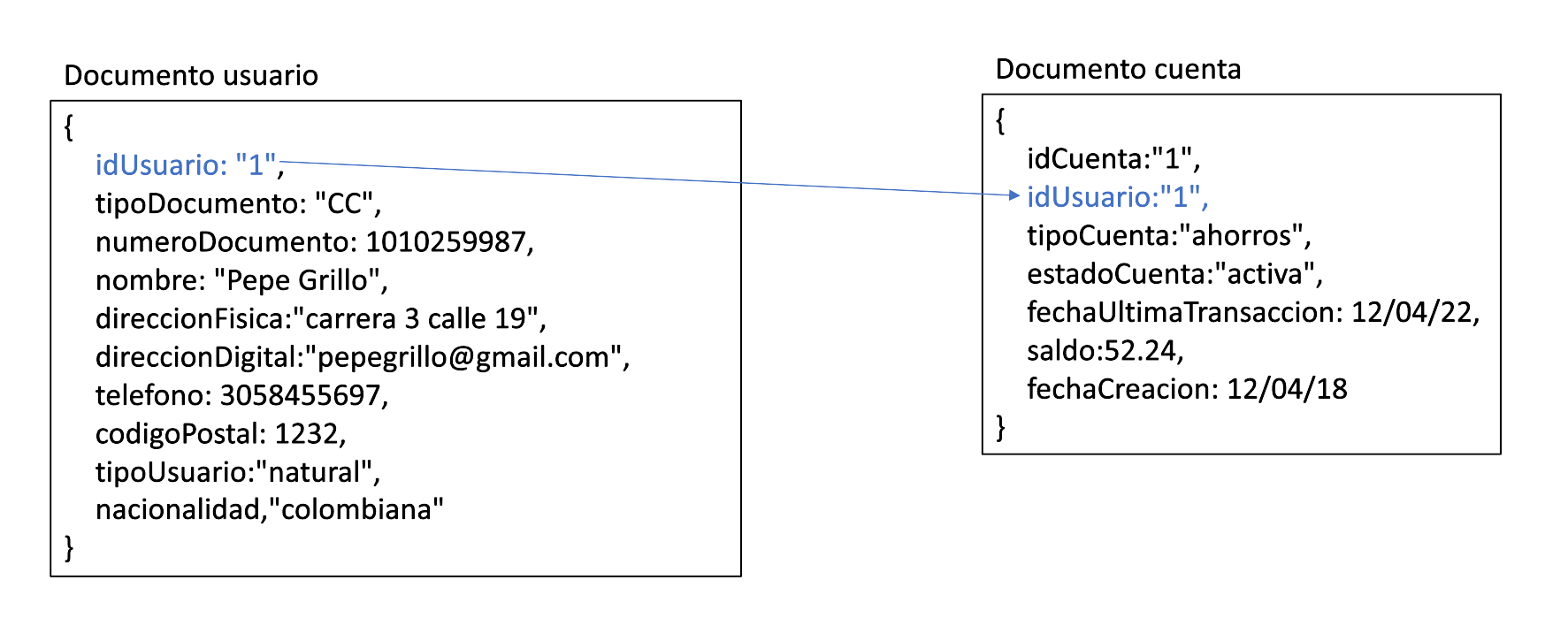
**Análisis:** Al responder a las preguntas, se observa que, debido a las particularidades de la búsqueda, actualización y simplicidad, la mejor solución es utilizar una referencia en esta relación. Por lo tanto, la clase Usuario tendría una referencia hacia la Oficina. Sin embargo, esta solución podría generar problemas con el paso de los años. Por ello, optaremos por utilizar una referencia inversa con el fin de evitar un arreglo que crezca indefinidamente.

Es importante destacar que comprometemos la eficiencia con esta elección, ya que una referencia directa sería más eficiente. Sin embargo, debido a las características del negocio, se sacrifica la escalabilidad y es necesario poder almacenar los usuarios, es decir, se necesita garantizar la permanencia. Además, una lista infinita de usuarios podría generar conflictos debido a las restricciones de memoria que puede tener un documento en MongoDB.

En conclusión, utilizar una referencia inversa para la relación entre oficinas y usuarios nos permitirá mantener un modelo de datos más manejable y optimizado para las consultas y actualizaciones, a pesar de los compromisos necesarios en términos de eficiencia y escalabilidad

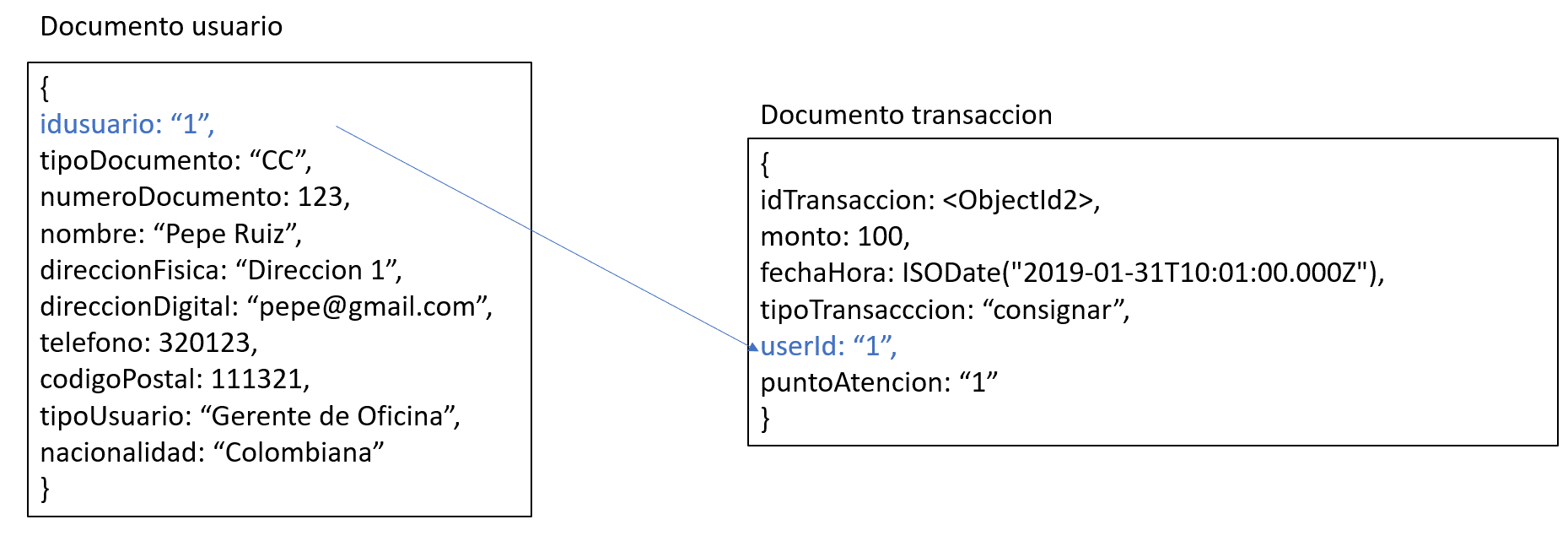
***Json de cada relación entre entidades***

* Usuario y cuenta: Referenciado



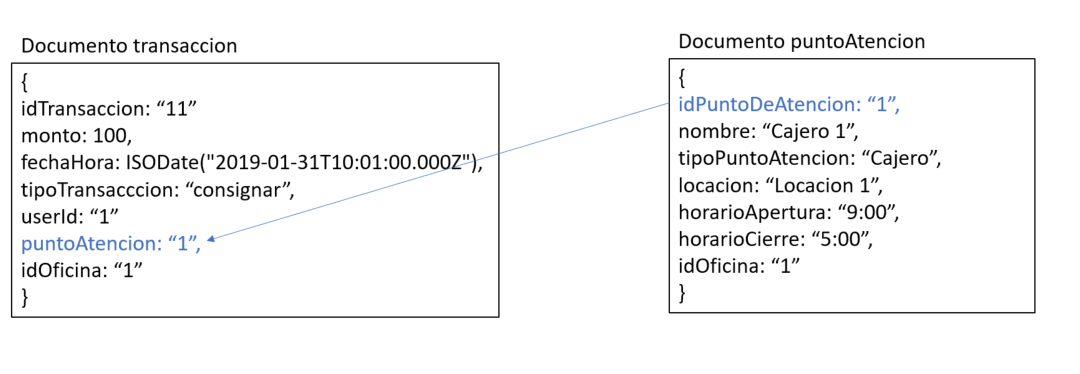
**Imagen 2. Diagrama Json Usuario y Cuenta**

* Usuario y transacción: Referenciado



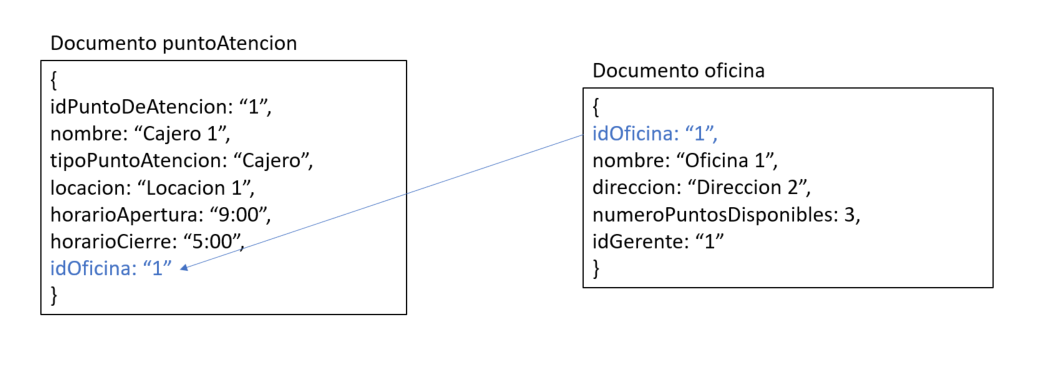
**Imagen 3. Diagrama Json Usuario y Transacción**

* Transacción y punto de atención: Referenciado



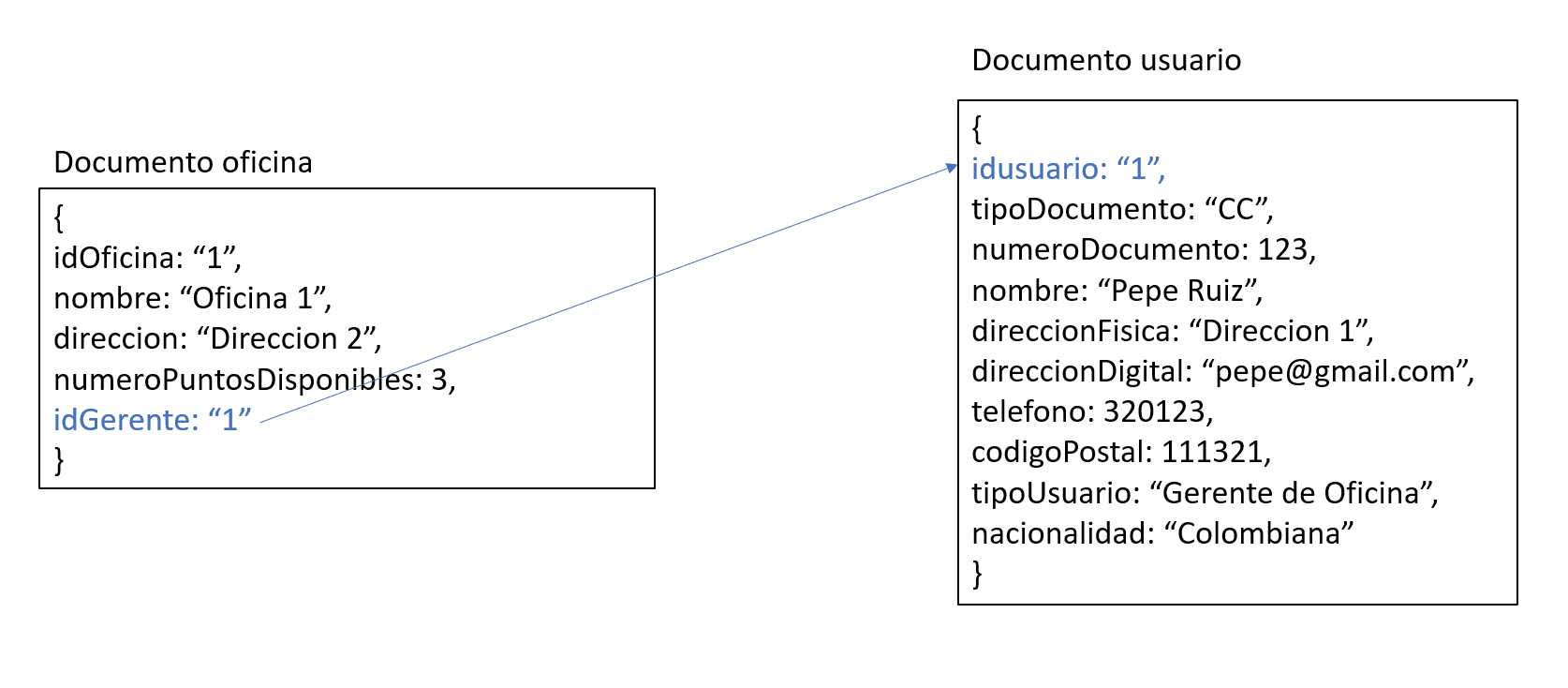
**Imagen 4. Diagrama Json Transacción y Punto De Atención**

* Punto de atención y oficina: Referenciado



**Imagen 5. Diagrama Json Punto de Atención y Oficina**

* Oficina y usuario: Referenciado



**Imagen 6. Diagrama Json Oficina y Usuario**

***Creación de colecciones en mongo***

Usuario:

db.createCollection("usuario", {

validator: {

$jsonSchema: {

bsonType: "object",

required: ["tipoDocumento", "numeroDocumento", "nombre", "direccionFisica", "direccionDigital", "telefono", "codigoPostal", "tipoUsuario", "nacionalidad"],

properties: {

tipoDocumento: { bsonType: "string" },

numeroDocumento: { bsonType: "int" },

nombre: { bsonType: "string" },

direccionFisica: { bsonType: "string" },

direccionDigital: { bsonType: "string" },

telefono: { bsonType: "int" },

codigoPostal: { bsonType: "int" },

tipoUsuario: { bsonType: "string" },

nacionalidad: { bsonType: "string" }

}

}

}

});

Transacción:

db.createCollection("transaccion", {

validator: {

$jsonSchema: {

bsonType: "object",

required: ["monto", "fechaHora", "tipoTransaccion", "userId", "puntoAtencionId"],

properties: {

monto: { bsonType: "double" },

fechaHora: { bsonType: "date" },

tipoTransaccion: { bsonType: "string" },

userId: { bsonType: "string" },

puntoAtencionId: {bsonType: "string"}

}

}

}

});

Punto de atención:

db.createCollection("puntoAtencion", {

validator: {

$jsonSchema: {

bsonType: "object",

required: ["nombre", "tipoPuntoAtencion", "horarioApertura", "horarioCierre"],

properties: {

nombre: { bsonType: "string" },

tipoPuntoAtencion: { bsonType: "string" },

locacion: { bsonType: "string" },

horarioApertura: { bsonType: "string" },

horarioCierre: { bsonType: "string" },

oficina: {bsonType: "string" }

}

}

}

});

Oficina:

db.createCollection("oficina", {

validator: {

$jsonSchema: {

bsonType: "object",

required: ["nombre", "direccion", "numeroPuntosDisponibles", "idGerente"],

properties: {

nombre: { bsonType: "string" },

direccion: { bsonType: "string" },

numeroPuntosDisponibles: { bsonType: "int" },

idGerente: { bsonType: "string" }

}

}

}

});

Cuenta:

db.createCollection("cuenta", {

validator: {

$jsonSchema: {

bsonType: "object",

required: ["idUsuario", "tipoCuenta", "estadoCuenta", "fechaUltimaTransaccion", "saldo", "fechaCreacion"],

properties: {

idUsuario: {bsonType: "string"},

tipoCuenta: { bsonType: "string" },

estadoCuenta: { bsonType: "string" },

fechaUltimaTransaccion: { bsonType: "date" },

saldo: { bsonType: "double" },

fechaCreacion: { bsonType: "date" }

}

}

}

});