

Modelado de Datos

Bases de Datos 2022

Esquemas Flexibles

- Por defecto los documentos de una colección pueden tener diferentes estructuras
- Cambio en la estructura implica actualizar los documentos a la nueva estructura
- En la práctica los documentos de una colección comparten una estructura similar
- Opcionalmente se pueden arreglar reglas de **validación de esquemas**

Validación de Esquemas

Validación de Esquemas

- Permite especificar reglas de validación a los documentos
- La validación de esquema se ejecuta durante los inserts y updates
- Cómo se especifican las reglas de validación
 - Especificación con [Validación de Esquemas JSON](#)
 - [Especificación con Operadores de Selección](#)

Validación de Esquemas JSON

```
db.createCollection( "<name>", { validator: <document>, validationLevel: <string>, validationAction: <string> } )
```

```
db.runCommand( { collMod: "<name>", validator: <document>, validationLevel: <string>, validationAction: <string> } )
```

➤ Crear la colección employees

```
db.createCollection("employees", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: [ "name", "age" ],
      properties: {
        name: {
          bsonType: "string",
          minLength: 3,
          description: "full name of the employee and is required"
        },
        age: {
          bsonType: "int",
          minimum: 16,
          description: "age of the employee and is required"
        },
        category: {
          enum: [ "Full-time", "Part-time", "Temporary" ],
          description: "can only be one of the enum values if the field exists"
        }
      }
    }
  }
})
```

➤ Ejemplo de validación 1

```
db.employees.insertOne( { name: "Kate MacDonell", category: "Full-time" } )
```

MongoServerError: Document failed validation

➤ Ejemplo de validación 2

```
db.employees.insertOne( { name: "Kate MacDonell", age: 15, category: "Part-time" } )
```

MongoServerError: Document failed validation

➤ Ejemplo de validación 3

```
db.employees.insertOne( { name: "Kate MacDonell", age: 21, category: "On-call" } )
```

MongoServerError: Document failed validation

➤ Ejemplo de validación 4

```
db.employees.insertOne( { name: "Kate MacDonell", age: 21, category: "Part-time" } )
```

```
{ acknowledged: true, insertedId: ObjectId("63648dff89f286435339acf3") }
```

Validación de Esquemas JSON - JSONSchema

```
{ $jsonSchema: <JSON Schema object> }
```

```
{  
  $jsonSchema: {  
    <keyword1>: <value1>,  
    <keyword2>: <value2>,  
    <keyword3>: <value3>,  
    ...  
  }  
}
```

Validación de Esquemas JSON - JSONSchema - KEYWORDS

➤ bsonType

- Acepta los mismos alias en string usados por el operador \$type

➤ required

- El documento debe contener todos los elementos especificados en el arreglo

➤ properties

- Un esquema JSON válido donde cada valor es un esquema JSON válido

➤ additionalProperties

- Especifica si campos si se permiten campos adicionales

➤ minimum, maximum

- Indica el valor mínimo (máximo) del campo

➤ minItems, maxItems

- Indica la longitud mínima (longitud máxima) del arreglo

➤ otros keywords

- [enum](#), [description](#), [pattern](#), [minLength](#), [maxLength](#), [uniqueItems](#), [más keywords](#)
- [Tips y buenas prácticas](#)

Validación de Esquemas con Operadores de Selección

- Permite especificar validaciones que comparan múltiples campos
- Se puede combinar validación con operador de selección con validación de esquemas JSON
- Crear la colección orders

```
db.createCollection( "orders",
{
  validator: {
    $expr: {
      $eq: [
        "$totalWithIVA",
        { $multiply: [ "$total", "$IVA" ] }
      ]
    }
  }
})
```

➤ Ejemplo de validación 1

```
db.orders.insertOne( {
  total: NumberDecimal("4000"),
  IVA: NumberDecimal("1.21"),
  totalWithIVA: NumberDecimal("4800")
})
```

MongoServerError: Document failed validation

➤ Ejemplo de validación 2

```
db.orders.insertOne( {
  total: NumberDecimal("4000"),
  IVA: NumberDecimal("1.21"),
  totalWithIVA: NumberDecimal("4840")
})
```

```
{ acknowledged: true, insertedId: ObjectId("6364977e89f286435339acf4") }
```


Validación de Esquemas - ValidationLevel - ValidationAction

- **validationLevel** : permiten especificar cómo aplicar las reglas de validación a documentos ya existentes
 - **strict**: (valor por defecto) Las reglas de validación se aplican a todos los inserts y updates.
 - **moderate** : Las reglas de validación solo se aplican a los documentos existentes válidos.
- **validationAction** : permiten especificar cómo manejar los documentos que no cumplen la validación
 - **error**: (valor por defecto) MongoDB rechaza cualquier insert o update que no cumple la regla de validación
 - **warn**: MongoDB permite que operación continúe, pero registra la infracción en los logs de MongoDB

Validación de Esquemas - ValidationLevel - ValidationAction

```
db.contacts.insertMany([
  { "_id": 1, "name": "Anne", "phone": "+1 555 123 456", "city": "London", "status": "Complete" },
  { "_id": 2, "name": "Ivan", "city": "Vancouver" }
])
```

➤ Agregar una validación a una colección existente

```
db.runCommand({
  collMod: "contacts",
  validator: { $jsonSchema: {
    bsonType: "object",
    required: [ "phone", "name" ],
    properties: {
      phone: {
        bsonType: "string",
        description: "phone must be a string and is required"
      },
      name: {
        bsonType: "string",
        description: "name must be a string and is required"
      }
    }
  }
},
  validationLevel: "moderate",
  validationAction: "error"
})
```

➤ Ejemplo de validación 1

```
db.contacts.updateOne(
  { "_id": 1 },
  { $set: { phone: null } }
)
```

MongoServerError: Document failed validation

➤ Ejemplo de validación 2

```
db.contacts.updateOne(
  { "_id": 2 },
  { $set: { phone: null } }
)
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Validación de Esquemas - Metadata

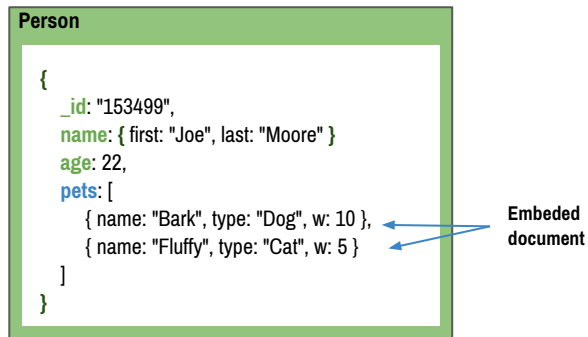
- Mostrar la regla de validación de una colección
 - `db.getCollectionInfos("<collection>")`

Modelado de Datos

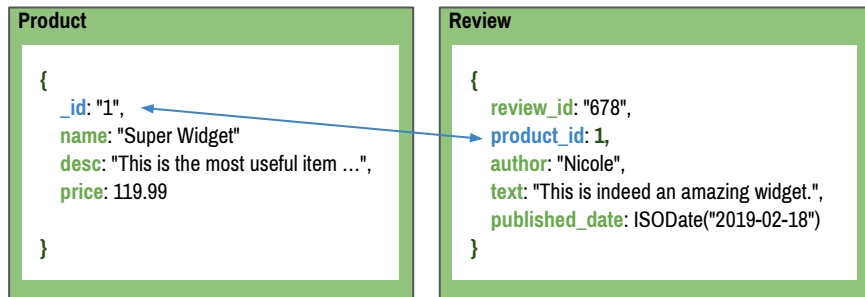
Estrategias de modelado de datos

- Decisión clave para el modelado de relaciones
 - Anidar (**embed**) datos VS. usar Referencias (**references**)

Modelo de datos Anidado



Modelo de datos usando Referencias



Modelado de Relaciones

➤ Relaciones One-to-One

- Modelado con Documentos Anidados
- Modelado con Referencias de Documentos

➤ Relaciones One-to-Many

- Modelado con Documentos Anidados
- Modelado con Referencias de Documentos

Modelado de Relaciones - One-To-One

➤ Modelado con Referencias de Documentos

```
db.student.insertOne(
{
  "_id": "jmoore",
  "name": "James Moore"
})

db.address.insertOne(
{
  "_id": "a1"
  "student_id": "jmoore"
  "street": "123 Sesame St",
  "city": "Anytown",
  "zip": "12345"
})
```

➤ Modelado con Documentos Anidados

```
db.student.insertOne(
{
  "_id": "jmoore",
  "name": "James Moore",
  "address": {
    "street": "123 Sesame St",
    "city": "Anytown",
    "zip": "12345"
  }
})
```

Modelado de Relaciones - One-To-Many

➤ Modelado con Referencias de Documentos

```
db.student.insertOne( {  
  "_id": "jmoore",  
  "name": "James Moore"  
})  
  
db.address.insertOne( {  
  "_id": "a1"  
  "student_id": "jmoore"  
  "street": "123 Sesame St",  
  "city": "Anytown",  
  "zip": "12345"  
})  
  
db.address.insertOne( {  
  "_id": "a2"  
  "student_id": "jmoore"  
  "street": "321 Some Other Street ",  
  "city": "Boston",  
  "zip": "45678"  
})
```

➤ Modelado con Documentos Anidados

```
db.student.insertOne(  
  {  
    "_id": "jmoore",  
    "name": "James Moore",  
    "address": [  
      {  
        "street": "123 Sesame St",  
        "city": "Anytown",  
        "zip": "12345"  
      },  
      {  
        "street": "321 Some Other Street ",  
        "city": "Boston",  
        "zip": "45678"  
      }  
    ]  
  }  
)
```


Modelado de Relaciones - One-To-Many - Referencias

➤ Modelado con arreglo de Referencias

```
db.parts.insertMany([ {
  "_id": "part1",
  partno: "123-aff-456",
  name: "#4 grommet",
  qty: 94,
  price: 3.99
}, {
  "_id": "partN",
  partno: "123-aff-678",
  name: "#5 grommet",
  qty: 94,
  price: 3.29
}])

db.products.insertOne({
  "_id": "product1",
  name: "left-handed smoke shifter",
  manufacturer: "Acme Corp",
  catalog_number: 1234,
  parts: [ "part1", "partN" ]
})
```

➤ Modelado con Referencias de Documentos

```
db.hosts.insertOne({
  "_id": "host1",
  name: "goofy.example.com",
  ipaddr: "127.66.66.66"
})

db.logmsg.insertMany([
  {
    _id: 1000001,
    time: ISODate("2014-03-28T09:42:41"),
    message: "cpu is on fire!",
    id_host: "host1"
  },
  {
    _id: 1000002,
    time: ISODate("2014-03-28T09:49:41"),
    message: "cpu is iddle!",
    id_host: "host1"
  }
])
```

Modelado de Datos Dirigido por Queries

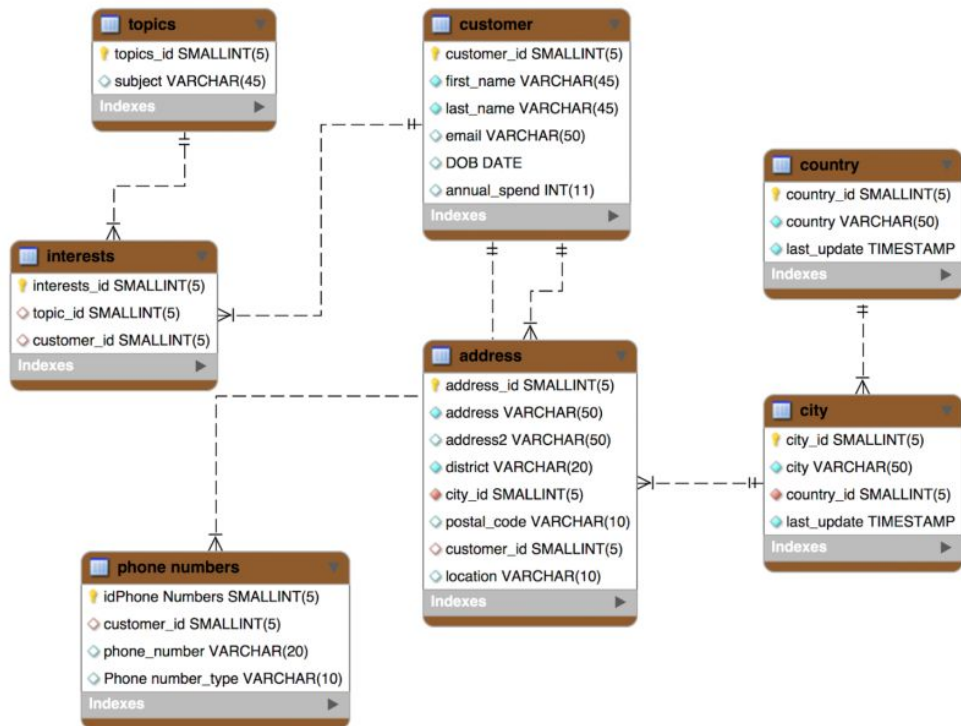
- Identificar entidades, atributos y relaciones
- Identificar las queries importantes
 - Y analizar la carga de trabajo de las queries (por ejemplo, frecuencia y latencia)
- Crear el modelo de datos aplicando las estrategias **Documento Anidados** y **Referencias**
 - Teniendo en mente las queries más importantes para crear el modelo de datos.
 - El modelo de datos debe permitir satisfacer las queries más importantes de manera eficiente
 - Una “consulta es eficiente” si se puede responder en un sola query sin \$lookup

IT'S DEMO TIME



Modelado de Datos Dirigido por Queries: Ejemplo

- Dado el diagrama junto con las queries más importantes crear un modelo datos en MongoDB



➤ Query 1:

Listar el id, nombre, apellido y teléfonos (número y tipo) de los clientes

➤ Query 2:

Listar los clientes (nombre, apellido y email) de una ciudad en particular

➤ Query 3:

Listar los clientes (nombre, apellido y email) interesados en un tópico en particular

Modelado de Datos Dirigido por Queries: Ejemplo

- Query 1: Listar el id, nombre, apellido y teléfonos (número y tipo) de los clientes
 - Entidades: customer y phone_numbers
 - Relación: One-To-Many
 - Estrategia: Documentos Anidados
- Query 2: Listar los clientes (nombre, apellido y email) de una ciudad en particular
 - Entidades: customer, address, city, y country
 - Relación: One-To-Many (entre customer y address)
 - Estrategia: Documentos Anidados
- Query 3: Listar los clientes (nombre, apellido y email) interesados en un tópico en particular
 - Entidades: customer, interests, topics
 - Relación: Many-To-Many (entre customer y topics)
 - Estrategia: Documentos Anidados

Modelado de Datos Dirigido por Queries: Ejemplo

➤ Modelo de datos en MongoDB

```
db.customer.insertOne( {  
  customer_id: "1",  
  name: { first: "John", last: "Moore" },  
  email: "jmoore@example.com",  
  annual_spend: 50000,  
  phone_numbers: [  
    {  
      type: "Home",  
      number: "238479823749"  
    }  
  ],  
  addresses: [  
    {  
      address: "sample address",  
      address2: "sample address2",  
      district: "sample district",  
      city: "sample city",  
      country: "sample country",  
      postal_code: "79878",  
      location: "sample location"  
    }  
  ],  
  topics: [ "topic 1", "topic 2" ]  
})
```

➤ Query 1

```
db.customer.find( { }, { customer_id: 1, name: 1, phone_numbers: 1, _id: 0 } )
```

➤ Query 2

```
db.customer.find(  
  { addresses: {  
    $elemMatch: { city: "sample city", country: "sample country"}  
  } },  
  { name: 1, email: 1, _id: 0 }  
)
```

➤ Query 3

```
db.customer.find( { topics: { $in: [ "topic 1" ] } }, { name: 1, email: 1, _id: 0 } )
```

Temas a estudiar

- Próxima clase
 - Indices
 - Replication y Sharding
 - Change Streams
- Referencias
 - [Validación de esquemas](#)
 - [Modelado de datos](#)