**Section 1: MONGODB OVERVIEW AND THE DOCUMENT MODEL (8%)**

1.1 Identify the set of value types MongoDB BSON supports.

BSON (Binary JSON) è il formato di dati binario utilizzato da MongoDB per rappresentare i documenti nei dati archiviati nel database. BSON supporta diversi tipi di valore. Ecco un elenco dei principali tipi di valore supportati da BSON in MongoDB:

1. **Double**: Numeri a virgola mobile a doppia precisione.
2. **String**: Stringhe di testo Unicode.
3. **Object**: Documenti incorporati BSON.
4. **Array**: Elenco ordinato di valori.
5. **Binary Data**: Dati binari, come immagini o file.
6. **Undefined**: Un valore speciale che indica un valore non definito.
7. **ObjectId**: Identificatore univoco a 12 byte per un documento.
8. **Boolean**: Valori booleani true o false.
9. **Date**: Data e orario.
10. **Null**: Un valore nullo.
11. **Regular Expression**: Espressione regolare con un modello e opzioni.
12. **JavaScript Code**: Codice JavaScript eseguibile.
13. **Symbol**: Simboli JavaScript (non comuni in MongoDB).
14. **JavaScript Code with Scope**: Codice JavaScript con un contesto (oggetti e variabili).
15. **32-bit Integer**: Numeri interi a 32 bit con segno.
16. **Timestamp**: Un timestamp consistente in un timestamp a 32 bit e un identificatore a 32 bit.
17. **64-bit Integer**: Numeri interi a 64 bit con segno.
18. **Decimal128**: Numeri decimali a 128 bit.
19. **Min Key**: Il valore più basso possibile.
20. **Max Key**: Il valore più alto possibile.

1.2 Given three documents that are of dierent shape, identify which can co-exist in the same collection.

**Section 2: CRUD (51%)**

2.1 Given a scenario with a type of structured document that needs to be inserted into a database, identify properly and improperly formed insert commands.

db.pizzas.insertOne({

"name": "Capricciosa",

"ingredients": ["tomato", "mozzarella", "ham", "mushrooms", "artichokes"],

"price": 12.99

})

2.2 Given an update scenario where an entire updated document (no update operators used) is provided, identify the output and how the database changed state.

Quando viene fornito un intero documento per un'operazione di aggiornamento senza l'utilizzo di operatori di aggiornamento, MongoDB sostituisce completamente il documento esistente con il nuovo documento fornito. Puoi farlo con il metodo replaceOne. Ecco un esempio:

db.pizzas.replaceOne( { type: "acciughe" }, {

\_id: ObjectId("657f2e00a2843d68d0faa3f6"),

name: "Acciughe",

ingredients: ["tomato", "mozzarella", "acciughe"],

price: 12.99

} )

**Output:**

Dopo aver eseguito questa operazione di aggiornamento, l'output sarà un oggetto UpdateResult con informazioni sull'operazione di aggiornamento, simile a quanto segue:

{

"acknowledged": true,

"matchedCount": 1, // Numero di documenti che corrispondono alla query

"modifiedCount": 1, // Numero di documenti effettivamente modificati (se diverso da matchedCount)

"upsertedId": null, // ID del documento upserted (se l'upsert è stato eseguito)

}

**Stato del database dopo l'aggiornamento:**

Dopo questa operazione di aggiornamento, lo stato del documento nella collezione sarà completamente sostituito dal nuovo documento fornito. Il documento originale è stato completamente sostituito dal nuovo documento.

**Scenari possibili:**

1. **Aggiornamento senza cambiamenti:** Se il nuovo documento è identico al documento esistente, il risultato mostrerà che matchedCount e modifiedCount sono entrambi 0, indicando che non ci sono stati effettivamente cambiamenti.
2. **Aggiornamento con cambiamenti:** Se il nuovo documento differisce dal documento esistente, il risultato mostrerà che matchedCount è 1 e modifiedCount è 1, indicando che c'è stato un documento corrispondente e che è stato effettivamente modificato

2.3 Given an update scenario where $set is used, identify the output and how the database changed state.

db.pizzas.updateOne( { type: "acciughe" }, { $set: { price: 11 } } )

2.4 Given a scenario about updating a document and information about where it should be inserted if it does not exist, identify the upsert command that should be used.

db.pizzas.updateMany( {"type": "marinara"}, { $set: { type: "marinara", price: 8 } }, { upsert: true } )

db.pizzas.updateOne( {"type": "acciughe"}, { $set: { type: "acciughe", price: 8 } }, { upsert: true } )

2.5 Given a scenario where multiple documents need to be updated, identify the correct update expression.

db.collectionName.updateMany(

{ "field": "value" }, // The condition to match documents

{ $set: { "newField": "newValue" } } // The update operation

)

2.6 Given a findAndModify scenario where another operation is run concurrently, identify the output and how the database changed state.

L'operazione findAndModify in MongoDB è una operazione atomica che consente di trovare, modificare e restituire un documento, il tutto in una singola operazione. Tuttavia, se un'altra operazione viene eseguita contemporaneamente, è importante comprendere come MongoDB gestisce queste situazioni.

Supponiamo di avere il seguente documento in una collezione:

{

"\_id": 1,

"name": "John",

"age": 30

}

E immaginiamo di eseguire una findAndModify per incrementare l'età di John di

const result = db.collectionName.findAndModify({

query: { "\_id": 1 },

update: { $inc: { "age": 1 } },

new: true // Restituisci il documento modificato

})

Ora, supponiamo che contemporaneamente un'altra operazione modifichi il documento utilizzando una query diversa, ad esempio:

db.collectionName.update(

{ "\_id": 1 },

{ $set: { "name": "Jane" } }

)

In questo caso, entrambe le operazioni vengono eseguite contemporaneamente. La sequenza degli eventi e l'output possono variare a seconda di quale delle due operazioni viene completata per prima.

### Scenario 1: findAndModify completato prima dell'update

1. findAndModify incrementa l'età di John di 1, restituendo il documento modificato.

{

"\_id": 1,

"name": "John",

"age": 31

}

* L'update viene applicato successivamente, ma non modifica l'età poiché il documento è già stato modificato.

{

"\_id": 1,

"name": "Jane",

"age": 31

}

### Scenario 2: update completato prima di findAndModify

1. update modifica il nome di John in Jane.

{

"\_id": 1,

"name": "Jane",

"age": 30

}

* findAndModify incrementa l'età di Jane di 1, restituendo il documento modificato.

{

"\_id": 1,

"name": "Jane",

"age": 31

}

2.7 Given a scenario where a document should be deleted from the database, identify the delete expression that should be used.

db.pizzas.deleteOne({\_id: 2})

2.8 Given a scenario where a single document should be looked up by a simple equality constraint (eg {x: 3}), identify the expression that should be used.

|  |
| --- |
| { |
| $lookup: |
| { |
| from: <collection to join>, |
| localField: <field from the input documents>, |
| foreignField: <field from the documents of the "from" collection>, |
| as: <output array field> |
| } |
| } |

[

{

$lookup: {

from: "accounts",

localField: "accounts",

foreignField: "account\_id",

as: "accounts\_customers"

}

}

]

2.9 Identify documents matched by a query with an equality constraint on an array field.

// Find pizzas where the "ingredients" array contains exactly ["tomato", "mozzarella"]

db.pizzas.find({ "ingredients": ["tomato", "mozzarella"] })

// Find pizzas where the "ingredients" array contains "tomato"

db.pizzas.find({ "ingredients": "tomato" })

// Find pizzas where the "ingredients" array contains "tomato"

db.pizzas.find({ "ingredients": "tomato" })

// Find pizzas where the "ingredients" array contains both "tomato" and "mozzarella"

db.pizzas.find({ "ingredients": { $all: ["tomato", "mozzarella"] } })

// Find pizzas where the "ingredients" array has an element that is an object with "name" equal to "tomato"

db.pizzas.find({ "ingredients": { $elemMatch: { $eq: "tomato" } } })

2.10 Identify documents matched by an expression with relational operators in it.

2.11 Identify documents matched by an expression with $in.

IL[$in](https://www.mongodb.com/docs/manual/reference/operator/query/in/" \l "mongodb-query-op.-in)l'operatore seleziona i documenti in cui il valore di un campo equivale a qualsiasi valore nell'array specificato.

db.pizzas.countDocuments( { price: { $in: [14.99, 19.99, 15.99] } } )

2.12 Identify documents matched by an $elemMatch expression.

db.pizzas.countDocuments( { ingredients: { $elemMatch: { $eq: "tomato", $eq: "mozzarella", $eq: "pineapple" } } } )

2.13 Identify documents matched by an expression that has several logical operators

|  |  |
| --- | --- |
| [$and](https://www.mongodb.com/docs/manual/reference/operator/query/and/" \l "mongodb-query-op.-and) | Joins query clauses with a logical AND returns all documents that match the conditions of both clauses. |
| [$not](https://www.mongodb.com/docs/manual/reference/operator/query/not/" \l "mongodb-query-op.-not) | Inverts the effect of a query expression and returns documents that do not match the query expression. |
| [$nor](https://www.mongodb.com/docs/manual/reference/operator/query/nor/" \l "mongodb-query-op.-nor) | Joins query clauses with a logical NOR returns all documents that fail to match both clauses. |
| [$or](https://www.mongodb.com/docs/manual/reference/operator/query/or/" \l "mongodb-query-op.-or) | Joins query clauses with a logical OR returns all documents that match the conditions of either clause. |

|  |
| --- |
| { $and: [ { <expression1> }, { <expression2> } , ... , { <expressionN> } ] } |

db.pizzas.find({

$and: [

{ "price": { $gte: 10, $lte: 15 } },

{ "ingredients": "tomato" },

{ "ingredients": "mozzarella" },

{ "ingredients": { $ne: "pineapple" } }

]

})

|  |
| --- |
| db.inventory.find( { |
| $and: [ |
| { $or: [ { qty: { $lt : 10 } }, { qty : { $gt: 50 } } ] }, |
| { $or: [ { sale: true }, { price : { $lt : 5 } } ] } |
| ] |
| } ) |

{ field: { $not: { <operator-expression> } } }

db.pizzas.find({

$and: [

{ "price": { $gt: 15 } }, // Prezzo superiore a $15

{

"ingredients": {

$not: {

$eq: "anchovies" // Non deve contenere l'ingrediente "anchovies"

}

}

}

]

})

|  |
| --- |
| { $nor: [ { <expression1> }, { <expression2> }, ... { <expressionN> } ] } |

{ $or: [ { <expression1> }, { <expression2> }, ... , { <expressionN> } ] }

**2.14** Given a query with a sort and limit, identify the correct output.

**db.pizzas.find().sort({type: 1, price:1}).limit(1)**

2.15 Identify the incorrect projection among a set of expressions.

2.16 Identify how to get all results from a cursor.

db.myCollection.find().toArray()

2.17 Identify the expressions used to count the number of documents matching a query.

db.restaurants.countDocuments({borough: "Brooklyn"})

2.18 Given an indexing scenario, identify the correct command for defining a search index.

db.restaurants.createIndex({ "name": "text", "description": "text" })

2.19 Given a scenario, identify the correct search query:

db.stores.find( { $text: { $search: **"java shop -coffee"** } } )

db.restaurants.aggregate([{ "$search": { "text": { "path": "name", "query": "carvel"} } }])

2.20 Given an aggregation expression using $match, $group, identify the correct output:

db.films.aggregate(

[

    {

        $match: {

            disponibile\_in\_4k: true

            }

    },

    {

        $group: {

            \_id: "$regista",

            sommaFilm: {

                $sum: 1

                }

            }

        }

    ]

)

2.21 Given an aggregation expression using $lookup, identify the correct output.

db.ordini.aggregate(

[

    {

        $lookup: {

            from: "utenti",

            localField: "utente\_id",

            foreignField: "\_id",

            as: "dettagli\_utente"

        }

    }

]

)

2.22 Given an aggregation expression using $out, identify the correct output:

db.films.aggregate(

[

    {

        $match: {

            disponibile\_in\_4k: true

            }

    },

    {

        $group: {

            \_id: "$regista",

            sommaFilm: {

                $sum: 1

                }

            }

        },

    {

        $out: {

            db:"prova",

            coll: "prova"

            }

        }

    ]

)