**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.

MongoDB utilizza una sintassi simile a JSON per inserire documenti in una collezione. I comandi di inserimento corretti in MongoDB coinvolgono l'uso del metodo insertOne con un documento JSON valido. I comandi di inserimento scorretti possono avere errori di sintassi o violare la struttura del documento di MongoDB.

**Comando di Inserimento Corretto:**

// Utilizzo del metodo insertOne per un singolo documento

db.nomeCollezione.insertOne({ chiave1: valore1, chiave2: valore2, ... })

**Comandi di Inserimento Scorretti:**

// Utilizzo scorretto delle virgole

db.nomeCollezione. insertOne ({ chiave1: valore1, chiave2: valore2, }) // Virgola extra alla fine

// Utilizzo di apici singoli al posto di doppi apici per le chiavi

db.nomeCollezione. insertOne ({ 'chiave1': valore1, 'chiave2': valore2 }) // Dovrebbero essere doppi apici per le chiavi

// Mancanza di un campo obbligatorio

db.nomeCollezione. insertOne ({ chiave1: valore1 }) // Se chiave2 è un campo obbligatorio, questo è scorretto

// Utilizzo di un nome di campo con un punto senza racchiuderlo tra virgolette

db.nomeCollezione. insertOne ({ nome.primo: 'John', nome.ultimo: 'Doe' }) // Dovrebbe essere { "nome.primo": "John", "nome.ultimo": "Doe" }

// Miscelazione di tipi di dati all'interno di un array

db.nomeCollezione. insertOne ({ chiave1: 'valore1', chiave2: 42, chiave3: [1, 'due', 3] }) // Tipi di dati non consistenti all'interno di chiave3

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/#_blank)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

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

            }

        }

    ]

)

**Section 3: INDEXES (17%)**

3.1 Given a query that is performing a collection scan, identify which index would improve the performance of this query.

//example query

db.listingsAndReviews.find().sort({beds: 1, bedrooms: 1, accomodates: 1})

//possibili index

db.listingsAndReviews.createIndex({ beds: 1, bedrooms: 1, accomodates: 1 });

db.listingsAndReviews.createIndex({ beds: 1, bedrooms: 1 });

db.listingsAndReviews.createIndex({ beds: 1 });

3.2 Given a query that is performing a collection scan on an equality match on an array field, identify which index would improve the performance of this query

//example query

db.customers.find({accounts: 987709})

Puoi creare un indice per un campo su cui stai eseguendo una query di uguaglianza. Ad esempio:

db.collection.createIndex({ campo: 1 })

db.collection.createIndex({ campo: -1 }).

3.3 Given a query with no constraint and a sort of two fields that is doing collection scan, identify which index would improve the performance of this query.

Per migliorare le prestazioni di una query senza vincoli e con un ordinamento su due campi che attualmente esegue una scansione completa della collezione, puoi creare un indice che includa entrambi i campi di ordinamento. Questo può velocizzare significativamente il processo di ordinamento. Ecco un esempio di come puoi creare un tale indice:

// Supponendo che tu abbia una collezione chiamata 'tuaCollezione'

db.tuaCollezione.createIndex({ campo1: 1, campo2: 1 });

In questo esempio:

campo1 e campo2 sono i campi sui quali stai ordinando.

I valori 1 nell'indice indicano l'ordinamento ascendente. Se vuoi ordinare in modo discendente, puoi utilizzare -1 al posto di 1.

Creando un indice su entrambi i campi coinvolti nell'ordinamento, MongoDB può utilizzare efficientemente questo indice per soddisfare i requisiti di ordinamento senza eseguire una scansione completa della collezione. Ciò è possibile perché l'indice contiene già le informazioni ordinate necessarie per la query.

Ricorda che la creazione di indici comporta un compromesso tra le prestazioni di lettura e quelle di scrittura, poiché gli indici devono essere aggiornati ad ogni operazione di scrittura. Pertanto, è importante considerare il tuo caso d'uso specifico e i modelli di query quando decidi quali indici creare.

3.4 Given a collection, identify how many indexes exist for that collection.

In MongoDB, è possibile utilizzare il metodo db.collection.getIndexes() per ottenere informazioni sugli indici esistenti per una specifica collezione. Questo metodo restituisce un array che contiene documenti con dettagli su ciascun indice.

3.5 Identify poor production practices.

Identificare pratiche di produzione insufficienti in un ambiente MongoDB implica cercare schemi o configurazioni che possono portare a prestazioni subottimali, vulnerabilità di sicurezza o problemi di manutenzione. Ecco alcune comuni pratiche di produzione insufficienti di cui essere consapevoli:

Assenza di Indicizzazione:

Non creare gli indici appropriati sui campi utilizzati nelle query può comportare prestazioni lente, specialmente su set di dati di grandi dimensioni.

Indicizzazione Inutile:

Creare indici non necessari, cioè indici che non vengono utilizzati, può influire negativamente sulle prestazioni di scrittura e comportare un consumo aggiuntivo di spazio di archiviazione.

Grande Numero di Piccoli Documenti:

Avere un grande numero di documenti di dimensioni ridotte può causare un aumento dell'overhead di archiviazione e avere un impatto negativo sulle prestazioni di lettura e scrittura.

Assenza di Sharding per Dataset Voluminosi:

Non utilizzare lo sharding per dataset di grandi dimensioni può limitare la scalabilità. Lo sharding contribuisce a distribuire i dati su più server per migliorare le prestazioni.

Query Inefficaci:

Eseguire query senza l'uso corretto degli indici o eseguire operazioni di aggregazione complesse in modo inefficiente può comportare colli di bottiglia nelle prestazioni.

Mancata Sicurezza del Database:

Negligenza nelle pratiche di sicurezza, come la mancata implementazione di autenticazione, autorizzazione o l'utilizzo di password deboli, può esporre il database a rischi per la sicurezza.

Assenza di Monitoraggio e Registrazione:

Non implementare adeguatamente pratiche di monitoraggio e registrazione rende difficile identificare e affrontare prontamente i problemi.

Assenza di Backup o Strategia di Backup Inadeguata:

Non eseguire regolarmente il backup dei dati o avere una strategia di backup inadeguata può comportare la perdita di dati in caso di guasti o disastri.

Ignorare la Configurazione di WiredTiger:

La configurazione di WiredTiger può influire sulle prestazioni. Ignorare le opzioni di ottimizzazione può portare a prestazioni subottimali.

Non Mantenere MongoDB Aggiornato:

Utilizzare versioni obsolete di MongoDB può esporre il sistema a vulnerabilità note e comportare la perdita di miglioramenti delle prestazioni e nuove funzionalità.

Utilizzo del Motore di Archiviazione MMAPv1:

MMAPv1 è un motore di archiviazione più vecchio in MongoDB, e utilizzarlo può causare problemi di prestazioni. In generale, si consiglia WiredTiger per le implementazioni moderne.

Ignorare il Write Concern e il Journaling:

Trascurare la configurazione corretta di write concern e journaling può comportare problemi di coerenza dei dati e potenziale perdita di dati.

Assenza di Connection Pooling:

Non utilizzare il connection pooling può comportare un uso inefficiente delle connessioni al database e influire sulle prestazioni complessive dell'applicazione.

Dimensione della Cache WiredTiger non Ottimizzata:

Non ottimizzare la dimensione della cache di WiredTiger può influire sull'efficienza delle operazioni di recupero e archiviazione dei dati.

È fondamentale rivedere e ottimizzare regolarmente le implementazioni MongoDB per garantire che rispettino le migliori pratiche e funzionino bene negli ambienti di produzione. Il monitoraggio regolare, la manutenzione e il mantenimento aggiornato della documentazione e degli aggiornamenti di MongoDB sono essenziali per mantenere una distribuzione sana e performante.

3.6 Identify the explain plan outputs that signify a potential performance issue, specifically whether an index is present or not for the given query.

Nell'analisi degli output del piano di spiegazione (explain plan) in MongoDB, ci sono diversi indicatori che possono segnalare potenziali problemi di prestazioni, in particolare in relazione alla presenza o assenza di un indice per la query fornita. Ecco alcuni componenti chiave del piano di spiegazione da considerare:

COLLSCAN (Scansione della Collezione):

Se vedi "stage" : "COLLSCAN" nei risultati di executionStats, indica che viene eseguita una scansione completa della collezione. Ciò significa che MongoDB sta esaminando l'intera collezione senza utilizzare un indice. Le scansioni di collezione sono generalmente meno efficienti rispetto agli scans dei dati tramite indice, specialmente per set di dati di grandi dimensioni.

IXSCAN (Scansione dell'Indice):

La presenza di "stage" : "IXSCAN" indica che viene utilizzata una scansione di indice. Questo è generalmente un segno positivo, poiché significa che MongoDB sta utilizzando un indice per recuperare i dati in modo più efficiente.

FETCH:

In uno stadio IXSCAN, se vedi "stage" : "FETCH", significa che l'ottimizzatore di query deve recuperare i documenti effettivi dalla collezione dopo aver utilizzato l'indice. Sebbene le scansioni degli indici siano efficienti, lo stadio di fetch può comunque comportare alcuni costi in termini di prestazioni.

SORT:

Se vedi "stage" : "SORT" nell'output di spiegazione, significa che viene eseguita un'operazione di ordinamento. L'ordinamento di grandi set di risultati può richiedere molte risorse ed è importante verificare se viene utilizzato un indice per l'ordinamento.

MEMORY\_LIMIT:

Se vedi "stage" : "MEMORY\_LIMIT" nel piano di spiegazione, potrebbe indicare che la query sta utilizzando una quantità eccessiva di memoria. Ciò potrebbe essere motivo di preoccupazione, specialmente se il sistema ha risorse limitate.

NESSUN INDICE:

L'assenza di un indice nella sezione winningPlan o executionStats potrebbe indicare un potenziale problema, specialmente se la query coinvolge filtri o ordinamenti. MongoDB potrebbe ricorrere a una scansione completa della collezione se non è disponibile un indice adeguato.

Ecco un esempio di cosa cercare nell'output di spiegazione:

"winningPlan" : {

"stage" : "COLLSCAN",

"direction" : "forward"

},

In questo esempio, "stage" : "COLLSCAN" indica una scansione completa della collezione.

Al contrario, un esempio di uno scenario positivo con l'utilizzo di un indice potrebbe apparire così:

"winningPlan" : {

"stage" : "IXSCAN",

"keyPattern" : {

"nomeCampo" : 1

},

"indexName" : "nomeCampo\_1"

},

In questo caso, "stage" : "IXSCAN" indica l'uso di un indice chiamato "nomeCampo\_1".

Comprendere e interpretare questi componenti nel piano di spiegazione può aiutare a identificare potenziali problemi di prestazioni e ottimizzare le query garantendo un uso efficace degli indici.

Indice Singolo:

Questo è l'indice più semplice e comune.

Viene creato su un singolo campo di un documento.

db.collection.createIndex({ field: 1 });

Indice Composto:

Questo indice coinvolge più campi e viene utilizzato per migliorare le query che coinvolgono più condizioni.

db.collection.createIndex({ field1: 1, field2: -1 });

Indice Testuale (Text Index):

Utilizzato per eseguire ricerche di testo completo su campi di stringa.

Può essere creato solo su campi di tipo stringa o array di stringhe.

db.collection.createIndex({ textField: "text" });

Indice Geospaziale:

Utilizzato per ottimizzare query che coinvolgono dati geospaziali.

Può essere creato su campi contenenti coordinate geografiche.

db.collection.createIndex({ locationField: "2dsphere" });

Indice Hashed:

Utilizzato per migliorare le prestazioni delle query di uguaglianza su campi di grandi dimensioni.

Crea un indice basato su hash del valore del campo.

db.collection.createIndex({ hashedField: "hashed" });

Indice Unico:

Garantisce che nessun documento nella collezione abbia valori duplicati per il campo indicizzato.

db.collection.createIndex({ uniqueField: 1 }, { unique: true });

Indice Parziale:

Si applica solo ai documenti che soddisfano una condizione specifica.

Utile per migliorare le prestazioni in caso di grandi collezioni con molte condizioni.

db.collection.createIndex(

{ partialIndexField: 1 },

{ partialFilterExpression: { status: "active" } }

);

Indice TTL (Time-To-Live):

Utilizzato per eliminare automaticamente i documenti dopo un certo periodo di tempo.

È basato su un campo che rappresenta una data di scadenza.

db.collection.createIndex({ expireField: 1 }, { expireAfterSeconds: 3600 });

Indice MultyKey:

Utilizzato per gli array:

db.collection.createIndex({ tags: 1 });

SEARCH INDEX:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**Section 4: DATA MODELING (4%)**

4.1 Given a scenario with three collections (a parent and two children) and the user, identify the embedded relationships and which should be linked.

4.2 Identify data model examples that are considered an anti-pattern.

1. [Massive arrays](https://developer.mongodb.com/article/schema-design-anti-pattern-massive-arrays?_ga=2.185272687.754950447.1703331471-1134864664.1699433974&_gac=1.251193076.1703410119.CjwKCAiAyp-sBhBSEiwAWWzTnsto525rUscMkX3fXMsoX8ILJDVySnejD8Vfvfp4daAH2uFt-rsmARoCL_YQAvD_BwE): :

**“I dati a cui si accede insieme dovrebbero essere archiviati insieme”, ma non necessariamente i dati correlati tra loro dovrebbero essere archiviati insieme.**

Quando incorporiamo documenti secondari correlati allo stesso documento, potremmo riscontrare un problema in cui la dimensione dell'array può aumentare e diventare un array illimitato. Ciò può causare i seguenti problemi:

I problemi:

* Limite dimensione documento 16 MB
* Le prestazioni dell'indice sull'array diminuiscono all'aumentare della dimensione dell'array

Soluzione:

Se vuoi eliminare il problema dell'array illimitato (a causa dell'incorporamento), devi utilizzare i riferimenti (Creazione di un nuovo schema).

1. [Massive number of collections](https://developer.mongodb.com/article/schema-design-anti-pattern-massive-number-collections):

WiredTiger è il motore di archiviazione predefinito di MongoDb, che crea file di indice per ogni raccolta. Se avremo un numero enorme di raccolte, ciò potrebbe causare un calo delle prestazioni poiché avremo più file indice. " **Il numero massimo consigliato di raccolte consentito è 10000** ", dopodiché potresti riscontrare un calo delle prestazioni.

1. [Unnecessary indexes](https://developer.mongodb.com/article/schema-design-anti-pattern-unnecessary-indexes):

Gli indici consentono a MongoDB di interrogare i dati in modo efficiente. Se una query non dispone di un indice per supportarla, MongoDB eseguirà una scansione delle raccolte per quella query. Ciò significa che ciascun documento dovrà recuperare i dati desiderati.

Perché non dovremmo creare indici non necessari?

* L'indice occupa spazio (Ogni indice occupa almeno 8 KB e cresce all'aumentare del numero di documenti ad esso associati)
* Gli indici influiscono sulle prestazioni del motore di archiviazione (numero enorme di raccolte, WiredTiger)
* Gli indici influiscono negativamente sulle prestazioni di scrittura (ogni volta che si verifica un'operazione CUD su uno o più documenti, vengono aggiornati anche gli indici collegati a tali documenti)

Indici da eliminare:

* Indici usati raramente
* Indici ridondanti

**Suggerimento:** limitare ciascuna raccolta a 50 indici

1. Documenti gonfi ([Bloated documents](https://developer.mongodb.com/article/schema-design-anti-pattern-bloated-documents?_ga=2.21786237.754950447.1703331471-1134864664.1699433974&_gac=1.147041093.1703410119.CjwKCAiAyp-sBhBSEiwAWWzTnsto525rUscMkX3fXMsoX8ILJDVySnejD8Vfvfp4daAH2uFt-rsmARoCL_YQAvD_BwE" \t "_blank)):

**“I dati a cui si accede insieme dovrebbero essere archiviati insieme”, ma non necessariamente i dati correlati tra loro dovrebbero essere archiviati insieme.**

Se avremo documenti di grandi dimensioni, avremo problemi durante la lettura dei dati. MongoDB utilizza la **cache** per recuperare i dati richiesti frequentemente anziché recuperare i dati dalla memoria WiredTiger. Quindi, anche se possiamo archiviare un documento con una dimensione massima di 16 MB, ciò creerà solo problemi durante il recupero dei dati come spiegato sopra.

Estrarre documenti da un disco è più lento che estrarli dalla cache.

Quanta **cache** è disponibile nella cache WiredTiger?

Il maggiore tra questi due numeri:

* 50% di (RAM-1 GB)
* 256 MB

Facciamo un esempio:

Se disponiamo di un cluster M10 con 2 GB di RAM, la cache otterrà la memoria di:

50% di (2 GB-1 GB) = 0,5 GB

Se il tuo set di lavoro (dati a cui si accede frequentemente) sarà inferiore o uguale a 0,5 GB, non è un problema. Se avremo una quantità di dati superiore a 0,5 GB, avrai due opzioni:

* Aggiorna il tuo cluster a M20 e così via (parlando di [MongoDB Atlas](https://cloud.mongodb.com/" \t "_blank) )
* Ristruttura il tuo database

La prima opzione è piuttosto semplice e devi solo aggiornare il tuo piano. La seconda opzione si basa su come è strutturato il tuo database e dipende dal tuo caso d'uso. Per eliminare i dati che non sono necessari frequentemente, è possibile spostarli in una nuova raccolta e utilizzare i riferimenti per collegarli.

La seconda opzione può portare alla duplicazione dei dati che presenta due inconvenienti:

1) Più spazio di archiviazione

2) Difficoltà nell'aggiornamento dei dati

Il primo problema può essere ignorato poiché lo spazio di archiviazione è economico. Il secondo problema può essere risolto ristrutturando i dati.

1. Query senza distinzione tra maiuscole e minuscole senza indici senza distinzione tra maiuscole e minuscole ([Case-insensitive queries without case-insensitive indexes](https://developer.mongodb.com/article/schema-design-anti-pattern-case-insensitive-query-index)):

Quando vogliamo interrogare i dati in cui dobbiamo abbinare alcune stringhe. Ad esempio, se disponiamo di un modulo per inserire il nome dell'animale, gli utenti possono inserire i nomi degli animali come di seguito per lo stesso animale Dog:

* Dog
* Dog
* dOg

Se abbiamo una query per recuperare i documenti con il nome dell'animale "Dog", allora possiamo usare l'espressione regolare per ottenere risultati senza distinzione tra maiuscole e minuscole utilizzando il parametro "i" durante l'esecuzione della query, ma non è così efficiente. Se utilizziamo regex, esaminerà tutti i documenti nella raccolta che causano problemi di prestazioni poiché regex non utilizza in modo efficiente gli indici senza distinzione tra maiuscole e minuscole. **Ma le query non regex non restituiscono i dati senza distinzione tra maiuscole e minuscole nella misura in cui utilizziamo un indice senza distinzione tra maiuscole e minuscole** .

Quindi ora, come possiamo creare un indice per cercare dati senza distinzione tra maiuscole e minuscole senza utilizzare la regex? Ecco la soluzione chiamata Collation.

Crea un indice di confronto che esamina solo i documenti che corrispondono alla query. Se abbiamo una query per cercare un animale Cane, questo esaminerà e restituirà solo i documenti con l'animale Cane senza scansionare tutti i documenti.

1. Separazione dei dati a cui si accede insieme ([Separating data that is accessed together](https://developer.mongodb.com/article/schema-design-anti-pattern-separating-data?_ga=2.150809200.754950447.1703331471-1134864664.1699433974&_gac=1.55807449.1703410119.CjwKCAiAyp-sBhBSEiwAWWzTnsto525rUscMkX3fXMsoX8ILJDVySnejD8Vfvfp4daAH2uFt-rsmARoCL_YQAvD_BwE" \t "_blank)):

**“I dati a cui si accede insieme dovrebbero essere archiviati insieme”, ma non necessariamente i dati correlati tra loro dovrebbero essere archiviati insieme.**

Questo è il punto più ripetuto quando parliamo di Anti-pattern. Facciamo un esempio qui. Se abbiamo uno schema per un prodotto e uno schema per le varianti di un prodotto.

**Schemi:**

* Prodotto
* Variante

Se sappiamo che una variante di un prodotto può essere riferimento a un solo prodotto e le varianti non crescono in modo esponenziale, allora possiamo eliminare lo schema Variant e inserire le varianti all'interno dello schema del prodotto ogni volta che otteniamo i dettagli del prodotto, dovremmo anche recuperare le varianti. Dopo aver risolto questo problema, avremmo un solo schema per il prodotto e le varianti incorporate nello schema del prodotto.

Riepilogo:

Se riassumiamo gli anti-pattern di Schema Design, la cosa fondamentale da tenere a mente è che **“i dati a cui si accede insieme dovrebbero essere archiviati insieme”.**Ma dobbiamo tenere a mente alcuni anti-modelli se seguiamo questa regola pratica. Non dovremmo sovraccaricare i documenti poiché abbiamo un limite di dimensione dei documenti di 16 MB e non dovremmo archiviare insieme i dati correlati, ma solo i dati a cui si accede insieme.

**Section 5: TOOLS AND TOOLING (2%)**

5.1 Given a scenario to load Atlas Sample Dataset and then use Data Explorer to use it to find a given first document in a collection

**Section 6: DRIVERS (18%)**

6.1 Define what the XX driver is?

Il **Java Driver per MongoDB** è una libreria che consente agli sviluppatori Java di comunicare con un database MongoDB. MongoDB è un sistema di gestione di database NoSQL che memorizza i dati in documenti JSON-like. Il Java Driver fornisce un'interfaccia di programmazione (API) che consente alle applicazioni Java di connettersi a un database MongoDB, eseguire operazioni come l'inserimento, l'aggiornamento, la cancellazione e l'interrogazione dei dati.

In sintesi, un Java driver è uno strumento che abilita l'interazione di un'applicazione scritta in Java con un particolare servizio o sistema, e il Java Driver for MongoDB è specificamente progettato per consentire alle applicazioni Java di lavorare con database MongoDB.

6.2 Define how the XX application connects/uses the XXX driver?

Per connettere e utilizzare il driver Java per MongoDB in un'applicazione Java, segui questi passaggi:

1. Aggiungi la dipendenza: Nel tuo progetto Java, aggiungi la dipendenza relativa al dri-ver Java per MongoDB.
2. Crea un'istanza del MongoClient: Nel tuo codice Java, crea un'istanza di MongoClient. Questo rappresenta la connessione al server MongoDB. Ad esempio:

MongoClient mongoClient = MongoClients.create("mongodb+srv://root:root@cluster0.32wvsiw.mongodb.net/?retryWrites=true&w=majority")

6.3 Define the components of the URI string used by MongoClient to connect the driver to the database.

Il componente URI utilizzato da MongoClient per connettersi a un database MongoDB include diverse parti essenziali:

1. Protocollo (mongodb://): Il protocollo indica il tipo di connessione. Per MongoDB, il protocollo è "mongodb://".
2. Nome utente e password (username:password@): Questa parte è opzionale e rappresenta le credenziali di accesso al database. Se il tuo database richiede l'autenticazione, devi fornire il nome utente e la password.
3. Indirizzo del server (localhost:27017): Indica l'indirizzo IP o il nome del server e la porta su cui il server MongoDB è in ascolto. Nel nostro esempio, "localhost" è l'indirizzo del server e "27017" è la porta predefinita di MongoDB.
4. Nome del database (/nome\_del\_database): Indica il nome del database a cui connettersi. Questa parte è separata dal resto dell'URI da uno slash ("/").
5. Parametri aggiuntivi (?parametro=valore): Questa parte è opzionale e può includere diversi parametri separati da "&". Ad esempio, potresti specificare il parametro "ssl" per connessioni sicure tramite SSL.

Un esempio completo di URI per connettersi a un database MongoDB potrebbe apparire così:

mongodb+srv://root:pippo@cluster0.32wvsiw.mongodb.net/?retryWrites=true&w=majority

Componenti:

1. Protocollo: mongodb+srv:// indica l'utilizzo del servizio DNS di connessione per MongoDB.
2. Credenziali: root:pippo rappresenta il nome utente e la password per l'autenticazione al database.
3. Host e Porta: cluster0.32wvsiw.mongodb.net è l'host del server MongoDB. Non è specificata alcuna porta, poiché l'uso di mongodb+srv:// implica che il sistema utilizzerà la porta predefinita di MongoDB.
4. Parametri: ?retryWrites=true&w=majority sono i parametri dell'URI. In questo caso, retryWrites=true indica di ritentare automaticamente le operazioni di scrittura dopo un fallimento, e w=majority specifica che almeno la maggioranza dei nodi deve confermare un'operazione di scrittura per essere considerata riuscita.

6.4 Identify what connection pooling is in terms of the driver and what advantages it oers.

**Un pool di connessioni è una cache di connessioni al database aperte e pronte all'uso, gestite dal driver. La tua applicazione può ottenere connessioni dal pool in modo trasparente, eseguire operazioni e restituire le connessioni al pool. I pool di connessioni sono thread-safe.**

Benefici di un Pool di Connessioni:

1. Riduzione della Latenza dell'Applicazione: Un pool di connessioni contribuisce a ridurre la latenza dell'applicazione, in quanto evita la necessità di creare nuove connessioni ogni volta che l'applicazione deve interagire con il database.
2. Creazione di Connessioni all'Avvio: Un pool di connessioni crea connessioni durante l'avvio dell'applicazione. Le applicazioni non devono restituire manualmente le connessioni al pool; al contrario, le connessioni tornano automaticamente al pool.
3. Attività e Disponibilità delle Connessioni: Alcune connessioni sono attive, mentre altre sono inattive ma disponibili. Se la tua applicazione richiede una connessione e ce n'è una disponibile nel pool, non è necessario creare una nuova connessione.

Creazione e Utilizzo di un Pool di Connessioni:

1. Utilizza un'istanza dell'oggetto MongoClient del tuo driver.
2. La maggior parte dei driver fornisce un oggetto di tipo MongoClient.
3. Usa un'istanza di MongoClient per ogni applicazione, a meno che l'applicazione si stia connettendo a molti cluster separati. Ogni istanza di MongoClient gestisce il proprio pool di connessioni verso il cluster o il nodo MongoDB specificato durante la creazione dell'oggetto MongoClient. Gli oggetti MongoClient sono thread-safe nella maggior parte dei driver.

Autenticazione

Per utilizzare un pool di connessioni con LDAP, consulta il comportamento del pool di connessioni LDAP.

Sharded Cluster Connection Pooling

I router mongos hanno pool di connessioni per ogni nodo nel cluster. La disponibilità di connessioni ai singoli nodi all'interno di un cluster sharded influisce sulla latenza. Le operazioni devono attendere che venga stabilita una connessione.

Impostazioni di Configurazione del Pool di Connessioni. Per configurare il pool di connessioni, imposta le opzioni:

1. tramite l'URI di MongoDB,
2. programmaticamente durante la creazione dell'istanza di MongoClient, o

nei file di configurazione del framework dell'applicazione.

6.5 Identify the correct syntax for the XX driver to insert one document and to insert many documents.

INSERT ONE:

Puoi inserire un singolo documento in una raccolta utilizzando il insertOne() metodo su un MongoCollection oggetto. Per inserire un documento, costruisci un Document oggetto che contenga i campi e i valori che desideri archiviare. Se chiami il insertOne()metodo su una raccolta che non esiste ancora, il server lo crea automaticamente per te.

Dopo un inserimento riuscito, insertOne()restituisce un'istanza di InsertOneResult. Puoi recuperare informazioni come il \_id campo del documento inserito chiamando il getInsertedId() metodo sull'istanza InsertOneResult.

Se l'operazione di inserimento fallisce, il driver solleva un'eccezione.

Immagine che contiene testo, schermata, software, Sistema operativo

Descrizione generata automaticamente

INSERT MANY:

È possibile inserire più documenti in una raccolta in un'unica operazione chiamando il insertMany()metodo su un MongoCollection oggetto. Per inserirli, aggiungi i tuoi Document oggetti a List e passali List come argomento a insertMany(). Se chiami il insertMany()metodo su una raccolta che non esiste ancora, il server lo crea per te.

Dopo l'inserimento riuscito, insertMany()restituisce un'istanza di InsertManyResult. Puoi recuperare informazioni come i \_id campi dei documenti inseriti chiamando il getInsertedIds() metodo sull'istanza InsertManyResult.

Se l'operazione di inserimento fallisce, il driver solleva un'eccezione.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

6.6 Identify the correct syntax for the XX driver to update one document and to update many documents.

UPDATE ONE:

È possibile aggiornare un singolo documento utilizzando il updateOne()metodo su un MongoCollection oggetto. Il metodo accetta un **filtro** che corrisponde al documento che desideri aggiornare e un'istruzione **di aggiornamento** che indica al driver come modificare il documento corrispondente. Il updateOne() metodo aggiorna solo il primo documento che corrisponde al filtro.

Per eseguire un aggiornamento con il updateOne() metodo, è necessario passare un filtro di query e un documento di aggiornamento. Il filtro di query specifica i criteri per cui il documento su cui eseguire l'aggiornamento e il documento di aggiornamento fornisce istruzioni su quali modifiche apportare ad esso.

Facoltativamente è possibile passare un'istanza di UpdateOptions al updateOne()metodo per specificare il comportamento del metodo. Ad esempio, se si imposta il upsert campo dell'oggetto UpdateOptions su true, l'operazione inserisce un nuovo documento dai campi sia nella query che nel documento di aggiornamento se nessun documento corrisponde al filtro della query.

Dopo l'esecuzione riuscita, il updateOne() metodo restituisce un'istanza di UpdateResult. Puoi recuperare informazioni come il numero di documenti modificati chiamando il getModifiedCount()metodo o il valore del \_id campo chiamando il getUpsertedId() metodo se specificato upsert(true)in UpdateOptions un'istanza.

Se l'operazione di aggiornamento fallisce, il driver solleva un'eccezione. Ad esempio, se provi a impostare un valore per il campo immutabile \_id nel tuo documento di aggiornamento, il metodo lancia un MongoWriteException messaggio con il messaggio:

|  |
| --- |
| Performing an update on the path '\_id' would modify the immutable field '\_id' |

Se il tuo documento di aggiornamento contiene una modifica che viola le regole dell'indice univoco, il metodo lancia un MongoWriteException messaggio di errore che dovrebbe assomigliare a questo:

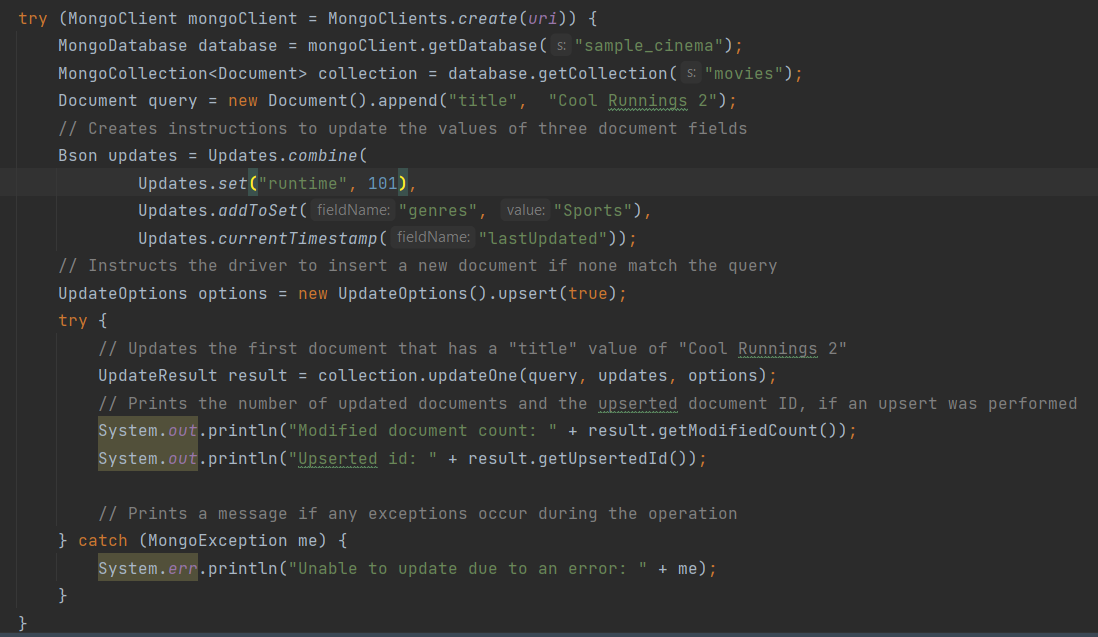
|  |
| --- |
| E11000 duplicate key error collection: ... |

## Esempio

In questo esempio, aggiorniamo la prima corrispondenza per la nostra query nella movies raccolta del sample\_cinema database. Eseguiamo i seguenti aggiornamenti al documento corrispondente:

* Imposta il valore di runtime a 99
* Aggiungi Sports all'array di genres solo se non esiste già
* Imposta il valore di lastUpdated sull'ora corrente.

Usiamo il Updates builder, una classe factory che contiene metodi di supporto statici, per costruire il documento di aggiornamento. Sebbene sia possibile passare un documento di aggiornamento invece di utilizzare il builder, il builder fornisce il controllo del tipo e una sintassi semplificata. Per ulteriori informazioni sul Updates builder, consulta la nostra [guida sul builder Aggiornamenti .](https://www.mongodb.com/docs/drivers/java/sync/current/fundamentals/builders/updates/)



UPDATE MANY:

È possibile aggiornare più documenti utilizzando il updateMany() metodo su un MongoCollection oggetto. Il metodo accetta un **filtro** che corrisponde al documento che desideri aggiornare e un'istruzione **di aggiornamento** che indica al driver come modificare il documento corrispondente. Il updateMany() metodo aggiorna tutti i documenti nella raccolta che corrispondono al filtro.

Per eseguire un aggiornamento con il updateMany() metodo, è necessario passare un filtro di query e un documento di aggiornamento. Il filtro di query specifica a quali documenti della raccolta corrispondere e il documento di aggiornamento fornisce istruzioni su quali modifiche apportare ad essi.

Facoltativamente è possibile passare un'istanza di UpdateOptions al  updateMany()metodo per modificare il comportamento della chiamata. Ad esempio, se si imposta il upsert campo dell'oggetto UpdateOptions su true e nessun documento corrisponde al filtro di query specificato, l'operazione inserisce un nuovo documento composto dai campi sia della query che del documento di aggiornamento.

Dopo l'esecuzione riuscita, il updateMany()metodo restituisce un'istanza di UpdateResult. È possibile recuperare informazioni come il numero di documenti modificati chiamando il getModifiedCount()metodo. Se hai specificato upsert (true) in un UpdateOptions oggetto e l'operazione risulta in un inserimento, puoi recuperare il \_id campo del nuovo documento chiamando il getUpsertedId()metodo sull'istanza UpdateResult.

Se l'operazione di aggiornamento fallisce, il driver solleva un'eccezione e non aggiorna nessuno dei documenti che corrispondono al filtro. Ad esempio, se provi a impostare un valore per il campo immutabile \_id nel documento di aggiornamento, il updateMany()metodo non aggiorna alcun documento e genera un MongoWriteExceptionmessaggio con il messaggio:

|  |
| --- |
| Performing an update on the path '\_id' would modify the immutable field '\_id' |

Se il tuo documento di aggiornamento contiene una modifica che viola le regole dell'indice univoco, il metodo lancia un MongoWriteException messaggio di errore che dovrebbe assomigliare a questo:

|  |
| --- |
| E11000 duplicate key error collection: ... |

## Esempio

In questo esempio, aggiorniamo i documenti che corrispondono alla nostra query nella movies  raccolta del sample\_cinema database. Eseguiamo i seguenti aggiornamenti ai documenti corrispondenti:

* Aggiungi Frequently Discussed all'array di genres solo se non esiste già
* Imposta il valore di lastUpdated sull'ora corrente.

Usiamo il Updatesbuilder, una classe factory che contiene metodi di supporto statici per costruire il documento di aggiornamento. Sebbene sia possibile passare un documento di aggiornamento invece di utilizzare il builder, il builder fornisce il controllo del tipo e una sintassi semplificata.

Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamente

REPLACE:

È possibile sostituire un singolo documento utilizzando il replaceOne() metodo su un MongoCollection oggetto. Questo metodo rimuove tutti i campi e i valori esistenti da un documento (eccetto il \_id campo) e lo sostituisce con il documento sostitutivo. Il replaceOne()metodo accetta un filtro di query che corrisponde al documento che desideri sostituire e un documento sostitutivo che contiene i dati che desideri salvare al posto del documento corrispondente. Il replaceOne() metodo sostituisce solo il primo documento che corrisponde al filtro. Facoltativamente è possibile passare un'istanza di ReplaceOptions al replaceOne() metodo per specificare il comportamento del metodo. Ad esempio, se si imposta il upsert campo dell'oggetto ReplaceOptions su true, l'operazione inserisce un nuovo documento dai campi nel documento sostitutivo se nessun documento corrisponde al filtro di query.  Dopo l'esecuzione riuscita, il replaceOne()metodo restituisce un'istanza di UpdateResult. È possibile recuperare informazioni come il numero di documenti modificati chiamando il getModifiedCount()metodo. Puoi anche recuperare il valore del \_id campo del documento chiamando il getUpsertedId()metodo se hai impostato upsert(true) l' ReplaceOptions istanza e l'operazione ha comportato l'inserimento di un nuovo documento. Se l'operazione di sostituzione fallisce, il driver solleva un'eccezione. Ad esempio, se provi a specificare un valore per il campo immutabile \_id nel documento sostitutivo che differisce dal documento originale, il metodo lancia un messaggio MongoWriteException con il messaggio:

|  |
| --- |
| After applying the update, the (immutable) field '\_id' was found to have been altered to \_id: ObjectId('...) |

Se il documento sostitutivo contiene una modifica che viola le regole dell'indice univoco, il metodo lancia un MongoWriteExceptionmessaggio di errore che dovrebbe assomigliare a questo:

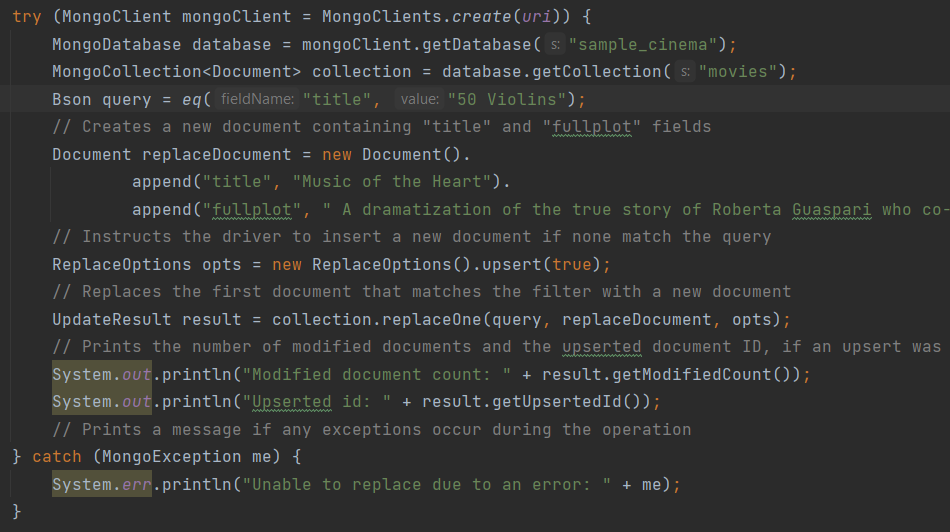
|  |
| --- |
| E11000 duplicate key error collection: ... |

## Esempio

In questo esempio, sostituiamo la prima corrispondenza del nostro filtro di query nella movies raccolta del sample\_cinema database con un documento sostitutivo. Tutti i campi tranne il \_id campo vengono eliminati dal documento originale e sostituiti dal documento sostitutivo. Prima dell'esecuzione replaceOne() dell'operazione, il documento originale contiene diversi campi che descrivono il film. Dopo l'esecuzione dell'operazione, il documento risultante contiene solo i campi specificati dal documento sostitutivo ( title e fullplot) e dal \_idcampo.

Il seguente frammento utilizza i seguenti oggetti e metodi:

* Un **filtro di query** passato al replaceOne()metodo. Il eq filtro corrisponde solo ai film il cui titolo corrisponde esattamente al testo 'Music of the Heart'.
* Un **documento sostitutivo** che contiene il documento che sostituisce il documento corrispondente, se esistente.
* Un oggetto **replaceOptions** con l' upsert opzione impostata su true. Questa opzione specifica che il metodo deve inserire i dati contenuti nel documento sostitutivo se il filtro di query non corrisponde ad alcun documento.



6.7 Identify the correct syntax for the XX driver to delete one document and to delete many documents.

DELETE ONE:

Puoi eliminare un singolo documento da una raccolta utilizzando il deleteOne() metodo su un MongoCollection oggetto. Il metodo accetta un filtro di query che corrisponde al documento che desideri eliminare. Se non specifichi un filtro, MongoDB corrisponde al primo documento nella raccolta. Il deleteOne()metodo elimina solo il primo documento corrispondente.

Questo metodo restituisce un'istanza DeleteResult che contiene informazioni tra cui il numero di documenti eliminati a seguito dell'operazione. Se l'operazione di eliminazione fallisce

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

DELETE MANY:

Puoi eliminare più documenti da una raccolta in un'unica operazione chiamando il deleteMany() metodo su un MongoCollection oggetto.

Per specificare quali documenti eliminare, passa un filtro di query che corrisponda ai documenti che desideri eliminare. Se fornisci un documento vuoto, MongoDB abbina tutti i documenti nella raccolta e li elimina. Sebbene sia possibile deleteMany() eliminare tutti i documenti in una raccolta, considera l'utilizzo di questo drop() metodo per ottenere prestazioni migliori.

In caso di eliminazione riuscita, questo metodo restituisce un'istanza di DeleteResult. È possibile recuperare informazioni come il numero di documenti eliminati chiamando il getDeletedCount() metodo sull'istanza DeleteResult. Se l'operazione di eliminazione fallisce, il driver solleva un'eccezione.

Immagine che contiene testo, schermata, software, Sistema operativo

Descrizione generata automaticamente

6.8 Identify the correct syntax for the XX driver to find many documents and to find one document.

FIND ONE:

È possibile recuperare un singolo documento in una raccolta concatenando i metodi find() e first() su un MongoCollection oggetto. Puoi passare un filtro di query al find() metodo per eseguire una query e restituire i documenti che corrispondono al filtro nella raccolta. Se non includi un filtro, MongoDB restituisce tutti i documenti nella raccolta. Il first() metodo restituisce il primo documento corrispondente.

È inoltre possibile concatenare altri metodi al find()metodo, ad esempio sort() quello che organizza i documenti corrispondenti in un ordine specificato e projection() che configura i campi inclusi nei documenti restituiti.

Il find()metodo restituisce un'istanza di FindIterable, una classe che offre diversi metodi per accedere, organizzare e analizzare i risultati. FindIterable eredita anche i metodi dalla sua classe genitore, MongoIterable come first().

Il first() metodo restituisce il primo documento dai risultati recuperati o null se non sono presenti risultati.

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

FIND MANY:

Puoi eseguire una query per più documenti in una raccolta chiamando il find() metodo su un MongoCollection oggetto. Passa un filtro di query al find() metodo per eseguire query e restituire documenti che corrispondono al filtro nella raccolta. Se non includi un filtro, MongoDB restituisce tutti i documenti nella raccolta.

È inoltre possibile concatenare metodi al find()metodo, ad esempio sort()quello che organizza i documenti corrispondenti in un ordine specificato e projection()che configura i campi inclusi nei documenti restituiti.

Il find()metodo restituisce un'istanza di FindIterable, una classe che offre diversi metodi per accedere, organizzare e analizzare i risultati. FindIterable eredita anche i metodi dalla sua classe genitore, MongoIterable che implementa l'interfaccia Java principale Iterable.

Puoi chiamare il iterator()metodo su cui MongoIterable restituisce MongoCursor un'istanza che puoi utilizzare per attraversare i risultati. È possibile chiamare metodi ad MongoCursor esempio hasNext()per verificare se esistono risultati aggiuntivi o next()per restituire il documento successivo nella raccolta. Se nessun documento corrisponde alla query, la chiamata hasNext() restituisce un risultato false e pertanto la chiamata next()genera un'eccezione.

Se si richiama next() l'iteratore dopo che ha restituito il risultato finale o quando non esistono risultati, viene generata un'eccezione di tipo java.util.NoSuchElementException. Utilizzare sempre hasNext()per verificare che esistano ulteriori risultati prima di chiamare next().

6.9 Identify the correct syntax for the XX driver to create an aggregation pipeline.

IL[pipeline di aggregazione](https://docs.mongodb.com/manual/core/aggregation-pipeline/) è un framework per l'aggregazione dei dati modellato sul concetto di pipeline di elaborazione dei dati, proprio come la "pipe" nella Shell Linux. I documenti entrano in una pipeline a più fasi che trasforma i documenti in risultati aggregati. È il modo più potente per lavorare con i tuoi dati in MongoDB. Ci consentirà di effettuare query avanzate come raggruppare documenti, manipolare array, rimodellare modelli di documenti, ecc.

Vediamo come possiamo sfruttare questa potenza utilizzando Java.

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Il driver MongoDB fornisce molti aiutanti per rendere il codice facile da scrivere e leggere.

Come puoi vedere, ho risolto questo problema con:

* UN [$match stage](https://docs.mongodb.com/manual/reference/operator/aggregation/match/) filtrare i miei documenti e mantenere solo il codice postale del Texas,
* UN [$group stage](https://docs.mongodb.com/manual/reference/operator/aggregation/group/) per raggruppare i miei codici postali nelle città,
* UN [$project stage](https://docs.mongodb.com/manual/reference/operator/aggregation/project/) per rinominare il campo \_id per city un output pulito (non obbligatorio ma sono di classe),
* UN [$sort stage](https://docs.mongodb.com/manual/reference/operator/aggregation/sort/) ordinare in base alla popolazione decrescente,
* UN [$limit stage](https://docs.mongodb.com/manual/reference/operator/aggregation/limit/) per mantenere solo le 5 città più popolate.

In MongoDB 4.2, ci sono 30 diverse fasi della pipeline di aggregazione che puoi utilizzare per manipolare i tuoi documenti. Se vuoi saperne di più ti incoraggio a seguire questo corso su MongoDB University:[M121: Il framework di aggregazione MongoDB](https://university.mongodb.com/courses/M121/about?_ga=2.88804317.754950447.1703331471-1134864664.1699433974&_gac=1.191598424.1703410119.CjwKCAiAyp-sBhBSEiwAWWzTnsto525rUscMkX3fXMsoX8ILJDVySnejD8Vfvfp4daAH2uFt-rsmARoCL_YQAvD_BwE).

6.10 Identify the dierent syntax for the XX driver when using the MongoDB Query Language (MQL) and when using the Aggregation Framework