MongoDB

Relazionale vs Non-Relazionale

Relazionale (SQL)

- Basato sulla teoria relazionale nata con le «Regole di Codd» (1985)
- Caratterizzato dall'applicazione della teoria E/R
  - che necessita di un processo di «normalizzazione» del dato per l'ottimizzazione delle performances

Non (solo) Relazionale (NO-SQL)

- Caratterizzato dalla possibilità di memorizzare «collezioni» di informazioni
- Sono disponibili diverse soluzioni che utilizzano tecniche diverse per gestire tipologie di dati diverse

Relazionale vs Non-Relazionale

Relazionale (SQL)

- Gestione implicita dell'integrità referenziale
  - Che garantisce l'identificabilità di un dato
  - Oltre all'aggiunta di constraints per la validazione dell'informazione

Non (solo) Relazionale (NO-SQL)

- Tipologie:
  - Document-data
  - Columnar (colum oriented)
  - Contenitori key-value
  - Document
  - Graph database

Relazionale vs Non-Relazionale

Relazionale (SQL)

- Lavora con dati «naturalmente» strutturati
- I collegamenti tra le entitità sono legati a «constraints»
- Ha alte capacità di indicizzazione che ottimizzano le risposte
- Supporta le transazioni
- Supporta table row-oriented
- Ha un linguaggio interno (SQL)

Non (solo) Relazionale (NO-SQL)

- Memorizza grosse quantità di dati con strutture poco complesse
- Semplifica scalabilità e flessibilità ai cambiamenti di business
- Utilizza opzioni «schema-free» o «schema-on-read»
- Gestisce dati non strutturati (Big Data)
- Sono document-oriented

Topics

Topics

- Installazione
- Replicazione, scalabilità e monitoraggio
- Operazioni di inserimento, rimozione e modifica
- Estrazione dei dati con find
- Indicizzazione
- Modellazione dell'informazione
- Ordinamento e paginazione
- Aggregazione
- Ottimizzazione delle performances
- HTTP API
- Accesso tramite
  - C#
  - Java
  - PHP
  - Python

Installazione

Tipologie di Distribuzioni

- MongoDB Atlas
- MongoDB Enterprise
- MongoDB Community Edition

MongoDB Atlas

- Piattaforma multi-cloud
  - Sicurezza configurata attraverso diverse funzionalita built-in
    - Gestione degli accessi con un sofisticato sistema role-based
    - Isolamento di rete
    - Autenticazione always-on
    - Crittografia dei dati at-rest
- Gestito tramite CLI dedicata o tools di terze parti

MongoDB Enterprise

- Edizione commerciale
  - Funzionalità aggiuntive
    - In-memory storage engine
      - Alte prestazioni e bassa latenza
    - Funzioni di gestione avanzata della sicurezza
      - Controllo degli accessi tramite LDAP e Kerberos
    - Data encryption
- Disponibile attraverso
  - Sottoscrizione MongoDB Enterprise Advanced
    - Assistenza e tools aggiuntivi
  - Disponibile gratuitamente per valutazione e sviluppo

MongoDB Community Edition

- Data model flessibile
  - Queries ad-hoc
  - Indicizzazione secondaria
  - Aggregazioni in real-time
- Disponibile attraverso
  - Servizio fully-managed con MongoDB Atlas on line (gratuito in cluster da 512 MB)
    - Auto-scaling
    - Istanze serverless
    - Ricerca full-text
    - Distribuzione tra regioni e clouds
  - Download locale
  - Kubernetes

Replicazione
Scalabilità
Monitoraggio

Scalabilità

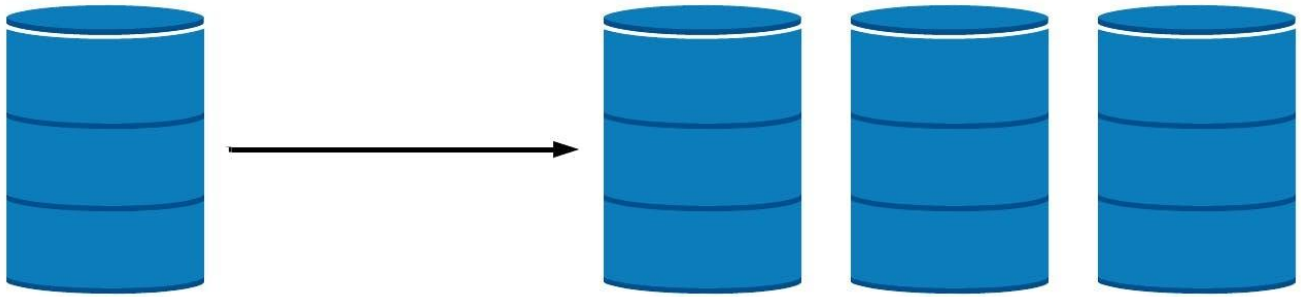
- La scalabilità del database è la capacità di espandere o contrarre la capacità delle risorse di sistema
  - al fine di supportare le modifiche alla logica applicativa
    - Che influenzano il suo uso crescente o decrescente
    - Un maggiore utilizzo dell'applicazione comporta tre sfide principali per il server del database:
      - Sovraccarico di lavoro della CPU o memoria che può causare latenza nella risposta
      - Esaurimento dell'area di storage, con conseguente impossibilità di memorizzare tutti i dati
      - Sovraccarico di lavoro nelle interfacce di rete, che impedisce di soddisfare le richieste degli utenti

Scalabilità

- Scalabilità verticale
  - Incremento di potenza di un singolo server
  - Sia i database relazionali che quelli non relazionali possono essere scalati, fino al limite fisico della macchina
- Scalabilità orizzontale
  - Duplicazione dei nodi per la condivisione delle risorse al fine di sgravare il compito dei server coinvolti distribuendolo su più macchine

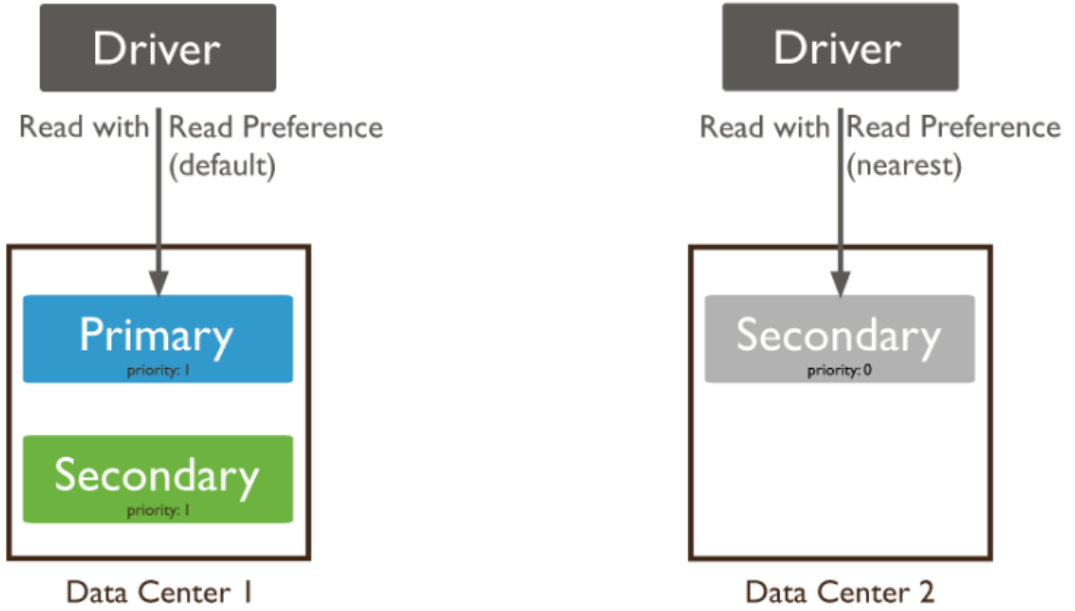
Replicazione

- Creazione di copie di un database o di un nodo di database
  - Con l'applicazione di un sistema di fault tolerance
  - Ogni nodo in un cluster contiene una copia dei dati
    - Se un nodo non può rispondere il cluster dirotta le richieste su altri nodi
- Rappresenta una forma di scalatura così i client possano essere distribuiti su tutti i nodi anziché operare su un nodo specifico
  - Questo incrementa la capacità del sistema di gestire più richeste



Replicazione

- In MongoDB un set di nodi replicati è chiamato «replica set»
  - In cui un nodo è definito «primario» e gli altri «secondari»
  - Le richieste sono distribuite su tutti i nodi
    - Ma solo sul primario si può scrivere così da consentire alle modifiche di essere replicate sugli altri nodi



Replicazione

- La replicazione è inclusa di default in MongoDB Atlas
- Non incrementa la capacità totale di storage
  - Né la capacità di gestire richieste di scrittura

Partizionamento (Sharding)

- Il «partizionamento» è la capacità di distribuire i dati sui nodi di un cluster
  - Ogni replica set in un cluster memorizza solo una porzione dei dati
    - Basata su una strategia key-sharding attraverso la quale si determina la distribuzione dei dati
  - Questo rende possibile aumentare la capacità di storage virtualmente senza limiti
  - Poiché ogni nodo gestisce dati propri, in esso possono essere effettuate operazioni sia di lettura che di scrittura

Partizionamento (Sharding)

Product	Price
Slipad	118
Muddus	88
Olle	37
Solsta	18
Tecken	60

Product	Price
Olle	37
Solsta	18

Product	Price
Muddus	88
Tecken	60

Product	Price
Slipad	118

Partizionamento (Sharding)

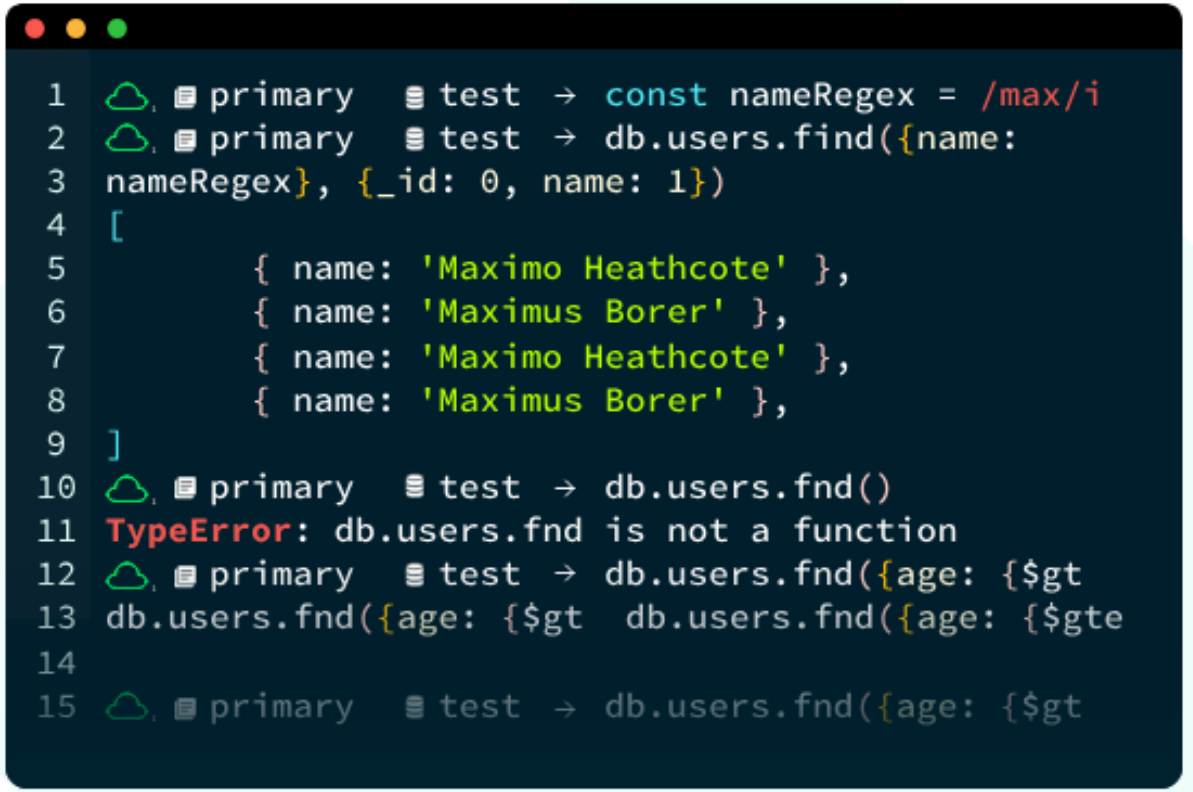
- Il partizionamento è una strategia di scalatura più complessa della replica
  - Poiché ogni nodo memorizza solo una parte dei dati, le queries hanno la necessità di determinare quali siano i nodi che contengono i dati rilevanti
  - L'applicazione client viene connessa ad un cluster sharded attraverso un router che inoltra le richieste verso il nodo appropriato
  - Se i dati sono contenuti su più nodi, le letture e scritture possono avvenire in parallelo
    - Con notevoli incrementi nelle performances

Mongo Shell e Compass

MongoDB Shell

- MongoDB Shell (mongosh)
  - Gestisce la connessione con server MongoDB consentendo di lavorare con i dati e di configurare il database
  - Prevede funzionalità avanzate che ne favoriscono l'usabilità:
    - Intelligent autocomplete
    - Syntax highliting
    - Help contestuali
    - Messaggi di errore semplici

MongoDB Shell



MongoDB Shell

- Connessione
  - mongosh

    "mongodb+srv://mycluster.abcd1.mongodb.net/myFirstDatabase"

    --apiVersion 1

    --username <username>

MongoDB Compass

- Potente GUI per MongoDB
- Tool interattivo per
  - Esecuzione di queries
    - Query bar per ricerche veloci
    - Data patterns discovery
  - Aggregazione dei dati
    - Builder per la creazione di potenti pipelines di aggregazione
  - Ottimizzazione ed analisi dei dati mantenuti su MongoDB
    - Gestione degli indici e controllo in real-time delle metriche del server
  - Drag & Drop per la creazione di «pipelines»

Basi

Formato BSON

- MongoDB memorizza i dati in un documento BSON
  - Rappresentazione binaria di un documento JSON

BSON - Data Types

- Double
- String
- Object
- Array
- Binary data
- ObjectId
- Boolean
- Date
- Null
- Regular Expression
- JavaScript
- 32-bit integer
- Timestamp
- 64-bit integer
- Decimal28
- Min key
- Max key

BSON - Confronti

- Nei confronti, MongoDB usa il seguente ordine (dal più basso al più alto)
  - MinKey, NULL, Numbers, String, Object, Array, Binary data, ObjectId, Boolean, Date, Timestamp, Regular expression, MaxKey
- Confronti tra dati dello stesso tipo
  - I dati numerici vengono convertiti al dominio più ampio
  - Le stringhe vengono confrontate in binario (salvo specifica di collation specifica)
  - Gli array vengono confrontati sulla base del tipo interno
  - Gli oggetti vengono confrontati sulla base delle coppie chiave-valore nell'ordine in cui appaiono
  - Le date vengono prima dei timestamps

Esecuzione di Comandi del DB

- Accesso al database
  - use <database_name>
- Esecuzione dei comandi
  - Oggetto db
    - Consente di accedere alle collezioni attraverso una notazione
      - db.collection_name
    - E quindi eseguire operazioni su di essa
    - La collezione viene creata se non esiste

Inserimento
Modifica
Cancellazione

Concetto di Collection e Document

- MongoDB memorizza i record come «document»
  - Anzi «BSON document»
- I documents sono organizzati in «collection»
- Un database memorizza una o più collection di document

Collections

- La creazione di una collection può avvenire
  - A seguito di un comando di inserimento dati
  - In maniera esplicita attraverso il metodo createCollection()
- Una collection non richiede la presenza di uno schema di validazione
  - Ma dalla versione 3.2 è possibile inserire delle regole per la validazione delle informazioni in inserimento e modifica

Collections


```
use myNewDB

db.myNewCollection1.insertOne( { x: 1 } )
```

```
db.myNewCollection2.insertOne( { x: 1 } )
db.myNewCollection3.createIndex( { y: 1 } )
```

```
db.createCollection(name, options)
```

Capped Collections

- Collezioni a dimensione fissa
  - Per consentire alte performances nell'inserimento e nel recupero quando basati sull'ordine di inserimento
  - Funzionano come un buffer circolare
    - In quanto, una volta terminato lo spazio a disposizione, i nuovi dati vanno a sovrascrivere quelli più vecchi

Capped Collections

- Comportamenti
  - L'ordine di inserimento è garantito
    - Le queries sono più veloci quando esse prevedono un ordinamento basato sull'ordine di inserimento
  - I documenti più vecchi vengono sovrascritti
    - Senza la necessità di ulteriori operazioni

Capped Collections

- Restrizioni
  - Non sono possibili snapshot
  - Per gli aggiornamenti è preferibile associare degli indici sui campi usati per il recupero degli elementi da aggiornare
  - Non è possibile lo sharding
  - Non è possibile scrivere su una capped collection in transazione

Clustered Collections

- Una collection sulla quale è creato un clustered index è detta clustered collection
- Vantaggi
  - Queries più veloci senza la necessità di indici quando l'ordinamento è quello di inserimento
  - Minore dimensione di storage
- Comportamenti
  - Un indice clustered può essere creato solo nel momento in cui si crea la collection
  - Le chiavi dell'indice sono memorizzate con la collection
- Limitazioni
  - Non si può trasformare una collection clustered in una non-clustered
  - L'indice clustered deve essere impostato sul campo _id
  - Non è possibile nascondere gli indici clustered
  - Una collection clustered non può essere capped

Schema Validation

- Contiene le informazioni per la verifica della correttezza (formale e sostanziale) delle informazioni presenti nei documenti
- Alla creazione di una collection può essere associato uno «schema validation document»
  - Attraverso il quale MongoDB verifica i dati in inserimento e modifica
- Lo schema può essere associato anche successivamente alla creazione
  - In tal caso
    - I nuovi documenti saranno validati in inserimento
    - I documenti già presenti nella collection non saranno validati fino alla loro prima modifica

SCHEMA Validation

- Quando la validazione fallisce
  - Per default MongoDB rifiuta l'operazione
  - Alternativamente il database può essere configurato per consentire comunque la modifica o l'inserimento con una registrazione delle violazioni all'interno dei files di log

Schema Validation

- Come si imposta lo schema di validazione
  - Attraverso la specifica di un JSON
  - Attraverso gli operatori di query
  - Attraverso la specifica dei valori consentiti per determinati campi

Schema Validation - JSON

```
db.createCollection("students", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         title: "Student Object Validation",
         required: [ "address", "major", "name", "year" ],
         properties: {
            name: {
               bsonType: "string",
               description: "'name' must be a string and is required"
            },
            year: {
               bsonType: "int",
               minimum: 2017,
               maximum: 3017,
               description: "'year' must be an integer in [ 2017, 3017 ]"
            },
            gpa: {
               bsonType: [ "double" ],
               description: "'gpa' must be a double if the field exists"
            }
         }
      }
   }
} )
```

```
MongoServerError: Document failed validation

Additional information: {
  failingDocumentId: ObjectId("630d093a931191850b40d0a9"),
  details: {
    operatorName: '$jsonSchema',
    title: 'Student Object Validation',
    schemaRulesNotSatisfied: [
      {
        operatorName: 'properties',
        propertiesNotSatisfied: [
          {
            propertyName: 'gpa',
            description: "'gpa' must be a double if the field exists",
            details: [
              {
                operatorName: 'bsonType',
                specifiedAs: { bsonType: [ 'double' ] },
                reason: 'type did not match',
                consideredValue: 3,
                consideredType: 'int'
              }
            ]
          }
        ]
      }
    ]
  }
}
```

Schema Validation - Query

```
db.createCollection( "orders",
   {
      validator: {
         $expr:
            {
               $eq: [
                  "$totalWithVAT",
                  { $multiply: [ "$total", { $sum:[ 1, "$VAT" ] } ] }
               ]
            }
      }
   }
)
```

Schema Validation – Field Values

```
db.createCollection("shipping", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         title: "Shipping Country Validation",
         properties: {
            country: {
               enum: [ "France", "United Kingdom", "United States" ],
               description: "Must be either France, United Kingdom, or
            }
         }
      }
   }
} )
```

Operazioni CRUD

- Create
- Read
- Update
- Delete

Create (Inserimento)

- Comandi
  - insertOne()
    - Per aggiungere un documento ad una collezione
  - insertMany()
    - Per aggiungere documenti multipli ad una collezione

Create (Inserimento)

```
use sample_mflix

db.movies.insertOne(
  {
    title: "The Favourite",
    genres: [ "Drama", "History" ],
    runtime: 121,
    rated: "R",
    year: 2018,
    directors: [ "Yorgos Lanthimos" ],
    cast: [ "Olivia Colman", "Emma Stone", "Rachel Weisz" ],
    type: "movie"
  }
)
```

```
use sample_mflix

db.movies.insertMany([
  {
    title: "Jurassic World: Fallen Kingdom",
    genres: [ "Action", "Sci-Fi" ],
    runtime: 130,
    rated: "PG-13",
    year: 2018,
    directors: [ "J. A. Bayona" ],
    cast: [ "Chris Pratt", "Bryce Dallas Howard", "Rafe Spall" ],
    type: "movie"
  },
  {
    title: "Tag",
    genres: [ "Comedy", "Action" ],
    runtime: 105,
    rated: "R",
    year: 2018,
    directors: [ "Jeff Tomsic" ],
    cast: [ "Annabelle Wallis", "Jeremy Renner", "Jon Hamm" ],
    type: "movie"
  }
])
```

Comportamenti di insert

- Se la collezione non esiste, l'operazione insert() la crea
- La chiave primaria, se di tipo ObjectId, viene creata automaticamente
- Tutte le operazioni di inserimento sono «atomiche» a livello di singolo documento
- Write Concern
  - Livello di confidenza richiesta da MongoDB per le operazioni di write

Insert - Opzioni

```
db.runCommand(
   {
      insert: <collection>,
      documents: [ <document>, <document>, <document>, ... ],
      ordered: <boolean>,
      maxTimeMS: <integer>,
      writeConcern: { <write concern> },
      bypassDocumentValidation: <boolean>,
      comment: <any>
   }
)
```

Insert Command

Campo	Tipo	Descrizione
insert	string	Il nome della collection
documents	array	Un array di documentida inserire nella collection
ordered	boolean	Determina se continuare l'inserimento in caso di errore (se false)
maxTimeMS	integer (positivo)	Limite di tempo in millisecondi (opzionale)
writeConcern	document	Un documento che determina il write concern di inserimento (opzionale)
bypassDocumentValidation	boolean	Disabilita i controlli di validazione (se true) (opzionale)
comment	any	Commento legato al comando (opzionale)

Insert Command

```
db.runCommand(
   {
      insert: "users",
      documents: [ { _id: 1, user: "abc123", status: "A" } ]
   }
)
```

```
{ "ok" : 1, "n" : 1 }
```

```
db.runCommand(
   {
      insert: "users",
      documents: [
         { _id: 2, user: "ijk123", status: "A" },
         { _id: 3, user: "xyz123", status: "p" },
         { _id: 4, user: "mop123", status: "p" }
      ],
      ordered: false,
      writeConcern: { w: "majority", wtimeout: 5000 }
   }
)
```

```
{ "ok" : 1, "n" : 3 }
```

Insert Command

```
db.createCollection("users", {
   validator:
      {
         status: {
            $in: [ "Unknown", "Incomplete" ]
         }
      }
})
```

```
db.runCommand({
      insert: "users",
      documents: [ {user: "123", status: "Active" } ]
})
```

```
db.runCommand({
   insert: "users",
   documents: [ {user: "123", status: "Active" } ],
   bypassDocumentValidation: true
})
```

```
{ "ok" : 1, "n" : 1 }
```

```
{
   n: 0,
   writeErrors: [
      {
         index: 0,
         code: 121,
         errInfo: {
            failingDocumentId: ObjectId('6197a7f2d84e85d1cc90d270'),
            details: {
               operatorName: '$in',
               specifiedAs: { status: { '$in': [Array] } },
               reason: 'no matching value found in array',
               consideredValue: 'Active'
            }
         },
         errmsg: 'Document failed validation'
      }
   ],
   ok: 1
}
```

Insert Command

- Output
  - insert.ok
    - The status of the command.
  - insert.n
    - The number of documents inserted.
  - insert.writeErrors
    - An array of documents that contains information regarding any error encountered during the insert operation. The
  - writeErrors
    - array contains an error document for each insert that errors.
    - Each error document contains the following fields:
    - insert.writeErrors.index
      - An integer that identifies the document in the documents array, which uses a zero-based index.
    - insert.writeErrors.code
      - An integer value identifying the error.
    - insert.writeErrors.errmsg
      - A description of the error.

Insert Command

- insert.writeConcernError
  - Document that describe error related to write concern and contains the field:
    - insert.writeConcernError.code
      - An integer value identifying the cause of the write concern error.
    - insert.writeConcernError.errmsg
      - A description of the cause of the write concern error.
    - insert.writeConcernError.errInfo.writeConcern
      - New in version 4.4.
  - The write concern object may also contain the following field, indicating the source of the write concern:
    - insert.writeConcernError.errInfo.writeConcern.provenance
      - A string value indicating where the write concern originated (known as write concern provenance). The following table shows the possible values for this field and their significance:
        - clientSupplied
          - The write concern was specified in the application.
        - customDefault
          - The write concern originated from a custom defined default value. See setDefaultRWConcern.
        - getLastErrorDefaults
          - The write concern originated from the replica set's settings.getLastErrorDefaults field.
        - implicitDefault
          - The write concern originated from the server in absence of all other write concern specifications.

Comando Update

- Serve a modificare i documenti all'interno di una collezione
- Un singolo comando può contenere molteplici istruzioni di update
- Il comando update mette a disposizione una serie di operatori per modificare i valori dei campi
  - Essi vanno passati come parametri al comando

Update (Modifica)

- Comandi:
  - updateOne()
  - updateMany()
  - replaceOne()
- Tutti i comandi accettano un oggetto che contiene operatori di update
  - $set -> per definire i valori da utilizzare per le modifiche
  - $*operator* -> usato per definire i criteri di ricerca degli elementi da modificare

Update (Modifica)

```
use sample_airbnb

db.listingsAndReviews.updateMany(
  { security_deposit: { $lt: 100 } },
  {
    $set: { security_deposit: 100, minimum_nights: 1 }
  }
)
```

```
use sample_mflix

db.movies.updateOne( { title: "Twilight" },
{
  $set: {
    plot: "A teenage girl risks everything-including her
  },
  $currentDate: { lastUpdated: true }
})
```

```
db.accounts.replaceOne(
  { account_id: 371138 },
  { account_id: 893421, limit: 5000, products: [ "Investment",
)
```

Operatori di Update per i campi

`$currentDate`	Sets the value of a field to current date, either as a Date or a Timestamp.
`$inc`	Increments the value of the field by the specified amount.
`$min`	Only updates the field if the specified value is less than the existing field value.
`$max`	Only updates the field if the specified value is greater than the existing field value.
`$mul`	Multiplies the value of the field by the specified amount.
`$rename`	Renames a field.
`$set`	Sets the value of a field in a document.
`$setOnInsert`	Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents.
`$unset`	Removes the specified field from a document.

Operatori di Update per gli Array

`$`	Acts as a placeholder to update the first element that matches the query condition.
`$[]`	Acts as a placeholder to update all elements in an array for the documents that match the query condition.
`$[<ident>]`	Acts as a placeholder to update all elements that match the arrayFilters condition for the documents that match the query condition.
`$addToSet`	Adds elements to an array only if they do not already exist in the set.
`$pop`	Removes the first or last item of an array.
`$pull`	Removes all array elements that match a specified query.
`$push`	Adds an item to an array.
`$pullAll`	Removes all matching values from an array.

Operatori di Update per Array Modificatori

- `$each` Modifies the $push and $addToSet operators to append multiple items for array updates.
- `$position` Modifies the $push operator to specify the position in the array to add elements.
- `$slice` Modifies the $push operator to limit the size of updated arrays.
- `$sort` Modifies the $push operator to reorder documents stored in an array.

Eliminazione

- Sono disponibili due metodi per la cancellazione di documenti in una collezione:
  - db.collection.deleteMany()
  - db.collection.deleteOne()
- Esempi:
  - Eliminazione di tutti i documenti: db.collection.deleteMany({})
  - Eliminazione di tutti i documenti che soddisfano una condizione:
    - db.collection.deleteMany({ field: "value"})
      - Cancella tutti i documenti in cui il campo «field» vale «value»
  - Eliminazione del primo documento che soddisfa una condizione:
    - db.collection.deleteOne({ field: "value"})
      - Cancella il «primo» documenti in cui il campo «field» vale «value»

Estrazione dei Dati

Esecuzione di Queries

- Per estrarre delle informazioni può essere usato il comando
  - db.collection.find()

```
use sample_mflix

db.movies.find()
```

SQL

```
SELECT * FROM movies
```

```
use sample_mflix

db.movies.find( { "title": "Titanic" } )
```

SQL

```
SELECT * FROM movies WHERE title = "Titanic"
```

```
use sample_mflix

db.movies.find( { rated: { $in: [ "PG", "PG-13" ] } } )
```

SQL

```
SELECT * FROM movies WHERE rated in ("PG", "PG-13")
```

Esecuzione di Queries

```
use sample_mflix

db.movies.find( { countries: "Mexico", "imdb.rating": { $gte: 7 } } )
```

AND

```
use sample_mflix

db.movies.find( {
     year: 2010,
     $or: [ { "awards.wins": { $gte: 5 } }, { genres: "Drama" } ]
} )
```

OR

Operatori di Query

- Sono disponibili diversi operatori:
  - Commenti
  - Bitwise
  - Array
  - Confronto
  - Logici
  - Basati su Elementi
  - Di Valutazione
  - Geospaziali

Operatori di Query

- Confronto:
  - Equals: $eq
  - Greater than: $gt
  - Less than: $lt
  - Greater or equal: $ge
  - Less or equal: $le
  - Not equal: $ne
  - Match value in array: $in
  - Match none value in array: $nin

Operatori di Query

- Logici:
  - AND: $and
  - OR: $or
  - NOR: $nor
  - NOT: $not

Operatori di Query

- Basati su singoli elementi:
  - Esistenza: $esists
  - Tipo di dato: $type
- Basati espressioni:
  - Validazione del documento: $jsonSchema
  - Confronto con un resto: $mod
  - Espressioni regolari: $regex
  - Basato su JavaScript: $where

Operatori di Query

- Operatori di array:
  - Confronto su tutti gli elementi: $all
  - Controllo delle dimensioni: $size
  - Confronto su una condizione basata su un elemento: $elemMatch

Operatori di Query

```
db.query_operator.find ()
db.query_operator.find ( { stud_id: { $eq: 1 } } )
db.query_operator.find ( { stud_id: { $gt: 2 } } )
db.query_operator.find ( { stud_id: { $lt: 2 } } )
db.query_operator.find ( {stud_id: { $lte: 2 } } )
db.query_operator.find ( { stud_id: { $gte: 2 } } )
```

```
db.query_operator.find ( { $and: [{stud_name: "ABC" }, { marks: {
$gte: 70 } } ] } )
db.query_operator.find ( { $or: [ { stud_id: 1 }, { stud_name: "ABC"
} ] } )
db.query_operator.find ( { $nor: [ { stud_id: 1 }, { stud_name: "ABC"
} ] } )
```

Operatori di Query

```
db.query_operator.find ( { "marks": { $exists: true, $gte: 50 } } )
db.query_operator.find ( { "marks": { $type: "double" } } )
```

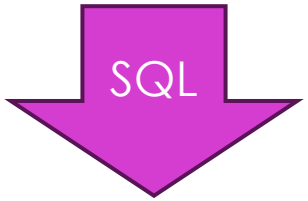
```
db.query_operator.find ( { "stud_name": { $regex: '.B.' } } )
```

```
db.query_operator.find ( { "marks": { $all: [ 60, 65 ] } } )
db.query_operator.find ( { "marks": { $size: 5 } } )
```

Proiezioni

- Una proiezione include nel risultato solo un insieme di campi
  - Metodo project()
    - Esempio: db.collection('name').find().project({ field_name: 1 })
  - È possibile sopprimere l'output del campo _id mettendolo esplicitamente a 0

```
db.inventory.find( { status: "A" }, { item: 1, status: 1 } )
```




```
SELECT _id, item, status from inventory WHERE status = "A"
```

Proiezioni

- Se non ci sono campi posti a 1, allora si intende voler includere tutti i campi ad esclusione di quelli posti a 0

```
db.inventory.find( { status: "A" }, { status: 0, instock: 0 } )
```

- È possibile riferirsi anche a sottodocumenti utilizzando la notazione puntata

```
db.inventory.find(
    { status: "A" },
    { item: 1, status: 1, "size.uom": 1 }
)
```

```
db.inventory.find(
    { status: "A" },
    { "size.uom": 0 }
)
```

```
{ size: { uom: 0 } }
```

Mongo 4.4

Proiezioni

- La notazione puntata può essere usata anche per proiettare campi in documenti interni ad un array
  - db.inventory.find({ status: "A" }, { item: 1, status: 1, "instock.qty": 1 })
  - db.inventory.find({ status: "A" }, { item: 1, status: 1, instock: { $slice: -1 } })

Indicizzazione

Indicizzazione

- Gli indici servono per rendere più efficiente l'esecuzione delle queries
  - Senza indici, MongoDB deve attraversare tutti i documenti per poter produrre i risultati di una query
- La scelta delle caratteristiche dell'indice è relativa alla specifica operazione, quindi va fatta dopo un'attenta analisi delle operazioni che intendiamo effettuare sul database
  - Alcuni editor di queries sono in grado di suggerire la necessità di inserimento di indici sui campi oggetto di indagine

Indicizzazione

- Creazione di un indice
  - createIndex()
    - In MongoDB Shell: db.collection.createIndex(<key and index type specification>, <options>)
  - Nelle specifiche dei campi coinvolti può essere utilizzato il valore 1 per un ordinamento crescent e -1 per un ordinamento decrescente
  - Il nome di un indice deve essere univoco

```
db.<collection>.createIndex(
   { <field>: <value> },
   { name: "<indexName>" }
)
```

```
db.blog.createIndex(
   {
     content: "text",
     "users.comments": "text",
     "users.profiles": "text"
   },
   {
     name: "InteractionsTextIndex"
   }
)
```

Indicizzazione

- Tipi di indici
  - Single Field Index
    - Colleziona e ordina i dati su un solo campo
      - Esempio: { userid: 1 }
  - Compound Index
    - Colleziona e ordina i dati sulla base di più campi
      - Esempio: { userid: 1, score: -1 }
  - Multikey Index
    - Colleziona e ordina i dati contenuti in array
    - Utile per migliorare le performance su campi all'interno di array

Indicizzazione

- Tipi di indici
  - Text Index
    - Indici ottimizzati per la ricerca full-text
    - Ogni collezione può avere un solo indice testuale
      - Che potrà coprire più campi

```
db.<collection>.createIndex(
   {
      <field1>: "text",
      <field2>: "text",
      ...
   }
)
```

Indicizzazione

- Tipi di indici
  - Wildcard Index
    - Poiché MongoDB supporta schemi «flessibili», un indice con wildcard consente di ottimizzare query su campi sconosciuti

```
db.collection.createIndex( { "$**": <sortOrder> } )
```

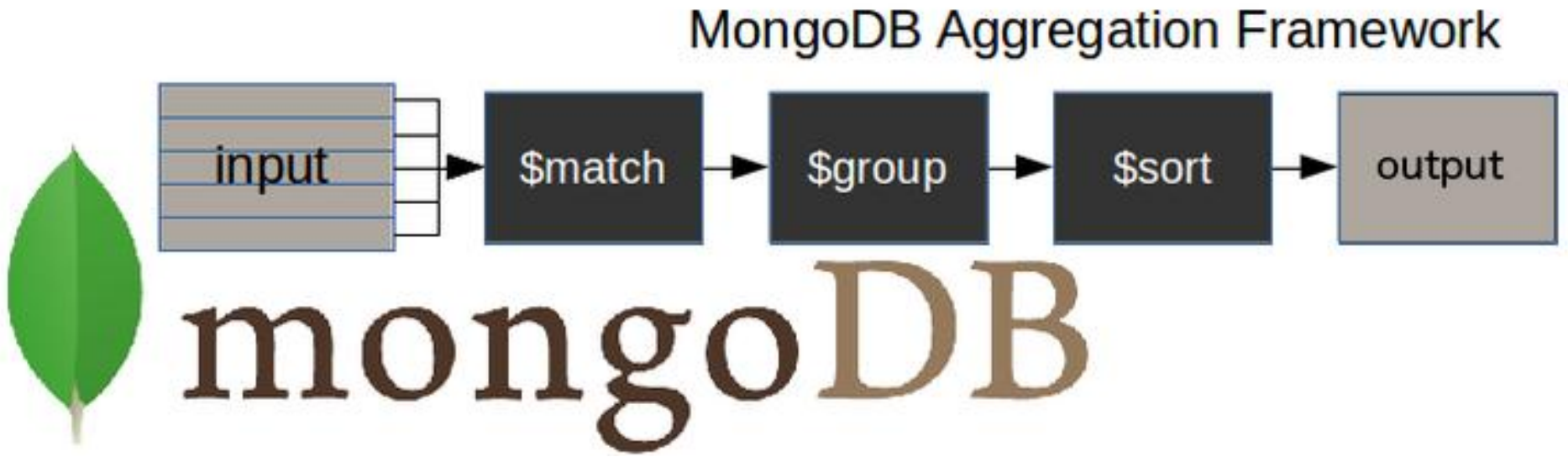
Indicizzazione

- Tipi di indici
  - Indici geospaziali
    - 2dsphere index
      - Per geometrie sferiche
    - 2d index
      - Per geomtrie piane

Indicizzazione

- Indice di default
  - MongoDB crea un indice univoco sul campo _id all'atto della creazione della collezione
    - Per impedire l'inserimento di due documenti con lo stesso _id
    - Non può essere cancellato
- Nomi degli indici
  - Il nome di default di un indice è dato dalla concatenazione dei campi coinvolti e dalla direzione delle chiavi in esso incluse
    - Un indice creato sui campi { item:1, quantity: -1} ha nome item_1_quantity_-1
  - Un indice non può essere rinominato (ma può essere cancellato e ricreato con altro nome)

Aggregazione


MongoDB Aggregation Framework
input
$match
$group
$sort
output
mongoDB

Aggregazione

- Operazione in cui vengono processati più documenti per ottenere dei risultati riassuntivi o di tendenza
  - Raggruppamenti con operazioni sui singoli gruppi
  - Analisi dei dati nel tempo

Aggregazione

- Pipeline
  - Metodo preferenziale in MongoDB
- Metodi di aggregazione per singoli usi
  - Più semplice ma meno potente e non riusabile

Aggregazione

- Aggregation Pipelines
  - Uno o più «stages» che processano i documenti:
    - Ogni stage esegue un'operazione su un documento di input
      - Es. filtro, raggruppamento, calcoli
    - L'output di uno stage diventa input dello stage successivo
  - Restituisce i risultati su gruppi di documenti

Aggregazione

Stage	Description
$addFields	Adds new fields to documents. Similar to $project, $addFields reshapes each document in the stream; specifically, by adding new fields to output documents that contain both the existing fields from the input documents and the newly added fields. $set is an alias for $addFields.
$bucket	Categorizes incoming documents into groups, called buckets, based on a specified expression and bucket boundaries.
$bucketAuto	Categorizes incoming documents into a specific number of groups, called buckets, based on a specified expression. Bucket boundaries are automatically determined in an attempt to evenly distribute the documents into the specified number of buckets.
$changeStream	Returns a Change Stream cursor for the collection. This stage can only occur once in an aggregation pipeline and it must occur as the first stage.
$changeStreamSplitLargeEvent	Splits large change stream events that exceed 16 MB into smaller fragments returned in a change stream cursor. You can only use $changeStreamSplitLargeEvent in a $changeStream pipeline and it must be the final stage in the pipeline.
$collStats	Returns statistics regarding a collection or view.
$count	Returns a count of the number of documents at this stage of the aggregation pipeline. Distinct from the $count aggregation accumulator.
$densify	Creates new documents in a sequence of documents where certain values in a field are missing.

Aggregazione

Stage	Description
$documents	Returns literal documents from input expressions.
$facet	Processes multiple aggregation pipelines within a single stage on the same set of input documents. Enables the creation of multi-faceted aggregations capable of characterizing data across multiple dimensions, or facets, in a single stage.
$fill	Populates null and missing field values within documents.
$geoNear	Returns an ordered stream of documents based on the proximity to a geospatial point. Incorporates the functionality of $match, $sort, and $limit for geospatial data. The output documents include an additional distance field and can include a location identifier field.
$graphLookup	Performs a recursive search on a collection. To each output document, adds a new array field that contains the traversal results of the recursive search for that document.
$group	Groups input documents by a specified identifier expression and applies the accumulator expression(s), if specified, to each group. Consumes all input documents and outputs one document per each distinct group. The output documents only contain the identifier field and, if specified, accumulated fields.
$indexStats	Returns statistics regarding the use of each index for the collection.
$limit	Passes the first n documents unmodified to the pipeline where n is the specified limit. For each input document, outputs either one document (for the first n documents) or zero documents (after the first n documents).

Aggregazione

Stage	Description
$listSampledQueries	Lists sampled queries for all collections or a specific collection.
$listSearchIndexes	Returns information about existing Atlas Search indexes on a specified collection.
$listSessions	Lists all sessions that have been active long enough to propagate to the system.sessions collection.
$lookup	Performs a left outer join to another collection in the same database to filter in documents from the "joined" collection for processing.
$match	Filters the document stream to allow only matching documents to pass unmodified into the next pipeline stage. $match uses standard MongoDB queries. For each input document, outputs either one document (a match) or zero documents (no match).
$merge	Writes the resulting documents of the aggregation pipeline to a collection. The stage can incorporate (insert new documents, merge documents, replace documents, keep existing documents, fail the operation, process documents with a custom update pipeline) the results into an output collection. To use the $merge stage, it must be the last stage in the pipeline. New in version 4.2.
$out	Writes the resulting documents of the aggregation pipeline to a collection. To use the $out stage, it must be the last stage in the pipeline.
$planCacheStats	Returns plan cache information for a collection.

Aggregazione

Stage	Description
$project	Reshapes each document in the stream, such as by adding new fields or removing existing fields. For each input document, outputs one document. See also $unset for removing existing fields.
$redact	Reshapes each document in the stream by restricting the content for each document based on information stored in the documents themselves. Incorporates the functionality of $project and $match. Can be used to implement field level redaction. For each input document, outputs either one or zero documents.
$replaceRoot	Replaces a document with the specified embedded document. The operation replaces all existing fields in the input document, including the _id field. Specify a document embedded in the input document to promote the embedded document to the top level. $replaceWith is an alias for $replaceRoot stage.
$replaceWith	Replaces a document with the specified embedded document. The operation replaces all existing fields in the input document, including the _id field. Specify a document embedded in the input document to promote the embedded document to the top level. $replaceWith is an alias for $replaceRoot stage.
$sample	Randomly selects the specified number of documents from its input.

Aggregazione

Stage	Description
$search	Performs a full-text search of the field or fields in an Atlas collection. NOTE $search is only available for MongoDB Atlas clusters, and is not available for self-managed deployments. To learn more, see Atlas Search Aggregation Pipeline Stages.
$searchMeta	Returns different types of metadata result documents for the Atlas Search query against an Atlas collection. NOTE $searchMeta is only available for MongoDB Atlas clusters running MongoDB v4.4.9 or higher, and is not available for self-managed deployments. To learn more, see Atlas Search Aggregation Pipeline Stages.
$set	Adds new fields to documents. Similar to $project, $set reshapes each document in the stream; specifically, by adding new fields to output documents that contain both the existing fields from the input documents and the newly added fields. $set is an alias for $addFields stage.

Aggregazione

Stage	Description
$setWindowFields	Groups documents into windows and applies one or more operators to the documents in each window. New in version 5.0.
$skip	Skips the first n documents where n is the specified skip number and passes the remaining documents unmodified to the pipeline. For each input document, outputs either zero documents (for the first n documents) or one document (if after the first n documents).
$sort	Reorders the document stream by a specified sort key. Only the order changes; the documents remain unmodified. For each input document, outputs one document.
$sortByCount	Groups incoming documents based on the value of a specified expression, then computes the count of documents in each distinct group.
$unionWith	Performs a union of two collections; i.e. combines pipeline results from two collections into a single result set. New in version 4.4.
$unset	Removes/excludes fields from documents. $unset is an alias for $project stage that removes fields.
$unwind	Deconstructs an array field from the input documents to output a document for each element. Each output document replaces the array with an element value. For each input document, outputs n documents where n is the number of array elements and can be zero for an empty array.

Serie Temporali

Serie Temporali

- Sequenze di dati analizzabili nel tempo
- Composte da:
  - Time
    - la data di registrazione del dato
  - Metadata (o source)
    - Una etichetta che identifica univocamente la serie (e non cambia, o cambia raramente)
  - Misure (o metriche o valori)
    - I dati tracciati al variare del tempo
      - Generalmente sono coppie chiave-valore

Serie Temporali

Example	Measurement	Metadata
Stock data	Stock price	Stock ticker, exchange
Weather data	Temperature	Sensor identifier, location
Website visitors	View count	URL

Serie Temporali

- Le collezioni basate su serie temporali
  - Sono collezioni che memorizzano efficientemente dati in serie temporali
  - Rispetto alle normali collezioni memorizzano i dati temporali in maniera pià performante
- MongoDB gestisce le Time Series Collections come viste scrivibili (non-materializzate) gestite da una collection interna
  - Che organizza i dati in serie temporali all'atto dell'inserimento del dato

Serie Temporali

- Creazione
  - Opzioni per la creazione della time-series
    - timeField: campo contenente il tempo
    - metaField: campo che contiene i metadati
    - granularity: intervallo gestito
    - expireAfterSeconds (opzionale):
      - Riferimento alla scadenza del documento quando il timeField supera una determinata durata

Serie Temporali

```
db.weather.insertMany( [
   {
      "metadata": { "sensorId": 5578, "type": "temperature" }
      "timestamp": ISODate("2021-05-18T00:00:00.000Z"),
      "temp": 12
   },
   {
      "metadata": { "sensorId": 5578, "type": "temperature" }
      "timestamp": ISODate("2021-05-18T04:00:00.000Z"),
      "temp": 11
   },
```

```
db.weather.findOne({
   "timestamp": ISODate("2021-05-18T00:00:00.000Z")
})
```

```
{
   timestamp: ISODate("2021-05-18T00:00:00.000Z"),
   metadata: { sensorId: 5578, type: 'temperature' },
   temp: 12,
   _id: ObjectId("62f11bbf1e52f124b84479ad")
}
```

Serie Temporali

```
db.weather.aggregate( [
    {
        $project: {
            date: {
                $dateToParts: { date: "$timestamp" }
            },
            temp: 1
        }
    },
    {
        $group: {
            _id: {
                date: {
                    year: "$date.year",
                    month: "$date.month",
                    day: "$date.day"
                }
            },
            avgTmp: { $avg: "$temp" }
        }
    }
] )
```

```
{
  "_id" : {
    "date" : {
      "year" : 2021,
      "month" : 5,
      "day" : 18
    }
  },
  "avgTmp" : 12.714285714285714
}
{
  "_id" : {
    "date" : {
      "year" : 2021,
      "month" : 5,
      "day" : 19
    }
  },
  "avgTmp" : 13
}
```

Data Modeling

Data Modeling

- Le strutture flessibili sono comode ma spesso è necessario prevedere strutture che semplifichino le operazioni (anche in considerazione delle ridondanze)
  - Dati embedded
    - Gestiscono le relazioni esterne tra i dati all'interno di un unico documento
    - I dati sono denormalizzati
  - References
    - I dati sono «normalizzati» alla stregua dei databases relazionali con l'utilizzo degli _id delle entities per gestire le chiavi esterne

Data Modeling e Transazioni

- Single Document Atomicity
  - Una operazione di write è atomica a livello di singolo documento
  - Dati denormalizzati possono prevedere aggiornamenti di entità parent e child in un contesto unico
- Multi-Document Transactions
  - Nel caso di aggiornamento di più documenti, le modifiche sono atomiche a livello di singolo documento
  - Dalla versione 4.2 è stato introdotto il concetto di transazione distribuita

Ottimizzazione

Ottimizzazione

- Query lente
  - Probabilmente si fa uno scan dell'intera collection invece di farlo su un indice
    - L'indice limita il numero di documenti da ispezionare
  - Una possibile soluzione è quella di aggiungere un indice
    - Questo implica sicuramente dei costi in scrittura e update
    - Troppi indici inutilizzati o sottoutilizzati possono rallentare modifiche e inserimenti
      - Ma nella maggior parte dei casi questo è un problema non rilevante

Ottimizzazioni

- Metriche di Cloud Manager
  - Scanned vs Returned
  - Scan and Order
  - WiredTiger Ticket Number
  - Document Structure Antipatterns
  - Unbounded arrays
  - Subdocuments without bounds

Ottimizzazioni

- Performance del database
  - Lag in replicazione
  - Decadimento di performance
  - Eccessivo uso di cursori
- Performance del cluster
  - Metriche di read e write
  - Metriche di hardware e network
  - Troppe connessioni client
  - Metriche di storage (esterno e memoria)

Data API

MongoDB Data API

- Set di endpoint HTTPS per estrazione e scrittura di dati
  - Configurazione
    - MongoDB dashboard → Data API → Enable Data API
    - Definizione del livello di accesso
      - No Access, Read Only, Read and Write, Custom
    - Creazione della chiave
- La richiesta deve essere inviata attraverso un client HTTP (curl)
  - Nel quale deve essere passata la chiave nella voce di header api-key

HTTP API

Esempio in C#

```
using MongoDB.Driver;
using Planets;
using System.Text;

var pwd = Encoding.UTF8.GetString(Convert.FromBase64String("cjF6ejAubmVsbDAubW9uZzA="));
var connectionString = "mongodb+srv://ennerre:r1zz0.nell0.mong0@cluster0.lh3sthm.mongodb.net/";
var dbName = "sample_guides";
var collectionName = "planets";

var client = new MongoClient(connectionString);
var database = client.GetDatabase(dbName);
var collection = database.GetCollection<Planet>(collectionName);

await collection.AsQueryable().ForEachAsync(Console.WriteLine);
```