LB165

Ellis Arn

Inhalt

[Anwendungsfall 2](#_Toc131874555)

[Handlungsziel 1 3](#_Toc131874556)

[Ich kann für einen Anwendungsfall eine geeignete NoSQL-Datenbank begründet auswählen und die Vorteile gegenüber einem anderen DBMS benennen. 3](#_Toc131874557)

[Ich kann die drei Eigenschaften des CAP-Theorems erläutern, die Eigenschaften am ausgewählten NoSQL-DMBS erläutern und die technische Umsetzung demonstrieren 3](#_Toc131874558)

[Handlungsziel 2 4](#_Toc131874559)

[Ich kann die Daten eines Anwendungsfalls für ein ausgewähltes NoSQL-DMBS strukturieren, das Datenmodell implementieren und Daten importieren. 4](#_Toc131874560)

[Strukturierung 4](#_Toc131874561)

[Importieren 5](#_Toc131874562)

[Datenmodell 10](#_Toc131874563)

[Ich kann Indices erstellen sowie Create-, Insert-, Update- und Delete-Operationen am Datenbestand einer NoSQL-Datenbank durchführen. 11](#_Toc131874564)

[Index 11](#_Toc131874565)

[CRUD 11](#_Toc131874566)

[Handlungsziel 3 14](#_Toc131874567)

[Ich definiere und begründe in Bezug auf den Anwendungsfall passende Zugriffsberechtigungen (Benutzende, Rollen) und implementiere diese in einem NoSQL-DBMS. 14](#_Toc131874568)

[Rollen 14](#_Toc131874569)

[Benutzer 15](#_Toc131874570)

[Handlungsziel 4 16](#_Toc131874571)

[Ich kann zwei Gründe für die Durchführung von Backups benennen. 16](#_Toc131874572)

[Ich kann ein Backup-Verfahren für eine NoSQL-DBMS implementieren und die Funktionsweise mit der Anwendung eines Wiederherstellungsverfahren belegen. 16](#_Toc131874573)

[Backup 16](#_Toc131874574)

[Löschen 16](#_Toc131874575)

[Restore 16](#_Toc131874576)

[Handlungsziel 5 17](#_Toc131874577)

[Ich kann für einen Anwendungsfall eine geeignete Replikationsart auswählen und erläutern, die Vorteile dieser Lösung im Vergleich zu Varianten aufzeigen und diese in Betrieb nehmen. 17](#_Toc131874578)

[Handlungsziel 6 18](#_Toc131874579)

[Ich kann Abfragen mit verschiedener Struktur (Selektion, Projektion) anwenden und dabei unterschiedliche Verfahren (Aggregation, parallele Verarbeitung) nutzen. 18](#_Toc131874580)

[Selektion und Projektion 18](#_Toc131874581)

[Aggregationen 19](#_Toc131874582)

[Ich kann eine Verbindung zwischen einer Programmiersprache und einem NoSQL-DBMS herstellen und CRUD-Operationen ausführen 21](#_Toc131874583)

# Anwendungsfall

Der Benutzer soll Spiele abfragen sollen und die Informationen zu den Spielen bekommen. Der Benutzer soll auch nach Tags, Genres, Entwicklern, usw. Daten abfragen tätigen (Z.B. den Durchschnittspreis eines bestimmten Genres oder von einem Entwickler). Bei den Daten handelt es sich um semistrukturierte Daten. Die Daten werden als einzelnes JSON-Dokument bereitgestellt und müssen vielleicht ein paar Mal im Jahr aktualisiert werden. Die Daten kommen aus der offiziellen Steam-API und von steamspy.com, werden aber im Voraus zusammengefügt aber nicht so bereinigt, dass man sie einfach einfügen kann. In der Datei sind über 55'000 Spiele und deren Beschreibung, Preis, Entwickler, usw. Die Aktualisierten Daten kommen ein paar Mal im Jahr in einem neuen separaten JSON-Dokument (nicht in der originalen Datei).

# Handlungsziel 1

## Ich kann für einen Anwendungsfall eine geeignete NoSQL-Datenbank begründet auswählen und die Vorteile gegenüber einem anderen DBMS benennen.

Eine geeignete NoSQL-Datenbank für den gegebenen Anwendungsfall wäre MongoDB. Die Vorteile von MongoDB gegenüber anderen DBMS wie relationalen Datenbanken sind:

* Hohe Skalierbarkeit und Flexibilität
* Einfache Handhabung von unstrukturierten und halbstrukturierten Daten
* Schnelle Lese- und Schreibzugriffe
* Dokumentenorientiertes Datenmodell

## Ich kann die drei Eigenschaften des CAP-Theorems erläutern, die Eigenschaften am ausgewählten NoSQL-DMBS erläutern und die technische Umsetzung demonstrieren

Das CAP-Theorem besagt, dass ein verteiltes Datenbanksystem nur zwei der drei folgenden Eigenschaften gleichzeitig gewährleisten kann: Konsistenz (C), Verfügbarkeit (A) und Partitionstoleranz (P).

* Konsistenz (C): Alle Knoten in der Datenbank sehen die gleichen Daten zur gleichen Zeit. Jede Leseanfrage erhält den neuesten Wert.
* Verfügbarkeit (A): Alle Knoten in der Datenbank sind immer verfügbar und können Anfragen beantworten.
* Partitionstoleranz (P): Das System kann weiterarbeiten, auch wenn die Kommunikation zwischen den Knoten unterbrochen ist.

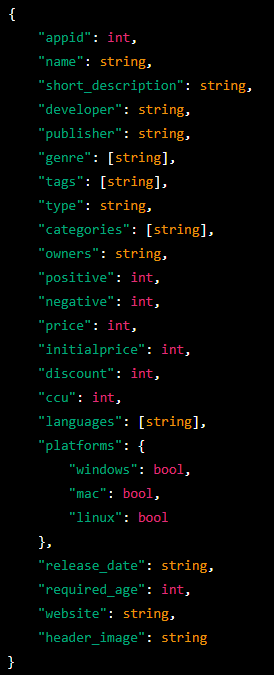
Hier am Beispiel von MongoDB mit Replikation (das, was ich gemacht habe):

MongoDB priorisiert Konsistenz (C) und Partitionstoleranz (P). In einer MongoDB-Replikatset-Architektur gibt es einen primären Knoten, der Schreibvorgänge akzeptiert, und sekundäre Knoten, die die Daten vom primären Knoten replizieren. Die Daten sind konsistent, da alle Schreibvorgänge auf dem primären Knoten durchgeführt werden und die sekundären Knoten die Daten vom primären Knoten replizieren. Wenn der primäre Knoten ausfällt, wird ein neuer primärer Knoten aus den verfügbaren sekundären Knoten gewählt, wodurch Partitionstoleranz erreicht wird.   
Verfügbarkeit kann unter bestimmten Umständen beeinträchtigt werden, beispielsweise wenn der primäre Knoten ausfällt und ein neuer primärer Knoten gewählt werden muss.  
Ausserdem kann durch Replikation auch eventual consistency geboten werden. Eventual consistency bedeutet, dass das System nach einer gewissen Zeit konsistent wird, aber nicht unbedingt sofort nach einer Schreiboperation. Diese Art von Konsistenz kann die Latenz und Verfügbarkeit in einem verteilten System verbessern, aber auf Kosten einer sofortigen Konsistenz.

# Handlungsziel 2

## Ich kann die Daten eines Anwendungsfalls für ein ausgewähltes NoSQL-DMBS strukturieren, das Datenmodell implementieren und Daten importieren.

### Datenmodell

Hier ein Bild meines Datenmodells:  


### Strukturierung

Ich habe in meinen Daten einiges geändert, damit sie besser zum Anwendungsfall passen und damit man sie besser abfragen kann. Ich habe sie so abgeändert, dass sie mit meinem Datenmodell übereinstimmen.

Ich habe mithilfe von regulären Ausdrücken (RegEx) Datensätze von Strings und Objekten in Arrays umgewandelt.

Hier habe ich die geschweifte Klammer bei «tags» in eine eckige Klammer umgewandelt.

|  |  |
| --- | --- |
| Suchen | ("tags":\s\*)\{ |
| Ersetzen | \1[ |

Hier habe ich das gleiche gemacht, nur dieses Mal für die zweite Klammer.

|  |  |
| --- | --- |
| Suchen | ("tags":\s\*\[[^\]]\*?)\}, |
| Ersetzen | \1], |

Hier habe ich ein Makro aufgezeichnet. Ich habe zuerst alles in «tags» ausgewählt und dann alle Doppelpunkte mit Zahlen in der Auswahl mit nichts ersetzt (gelöscht).

|  |  |
| --- | --- |
| Auswählen | ("tags"\s\*:\s\*\[([\s\S]\*?)\]) |
| Suchen | :\s\*\d+ |
| Ersetzen | Nichts |

Hier habe ich wieder ein Makro aufgezeichnet, welches alles in «languages» auswählt und dann alle Kommas, die ein Leerzeichen danach haben und sich in der Auswahl befanden mit ", " ersetzt. (Ich habe davor und danach Anführungszeichen gemacht)

|  |  |
| --- | --- |
| Auswählen | "languages":\s\*"(.+?)" |
| Suchen | , |
| Ersetzen | ", " |

Dann habe ich mit diesen Ausdrücken die Werte in «languages» mit eckigen Klammern umgeben.

|  |  |
| --- | --- |
| Suchen | "languages":\s\*"(.+?)" |
| Ersetzen | "languages": ["$1"] |

Hier habe ich das gleiche gemacht, nur dieses Mal bei «genre».

|  |  |
| --- | --- |
| Suchen | "genre":\s\*"([^"]+)" |
| Ersetzen | "genre": ["$1"] |

Hier habe ich wieder ein Makro gemacht, welches alles in «genre» auswählt und dann alle Kommas, die ein Leerzeichen danach haben und sich in der Auswahl befinden mit ", " ersetzt. (Ich habe davor und danach Anführungszeichen gemacht)

|  |  |
| --- | --- |
| Auswählen | "genre":\s\*\[([^\]]+)\] |
| Suchen | , |
| Ersetzen | ", " |

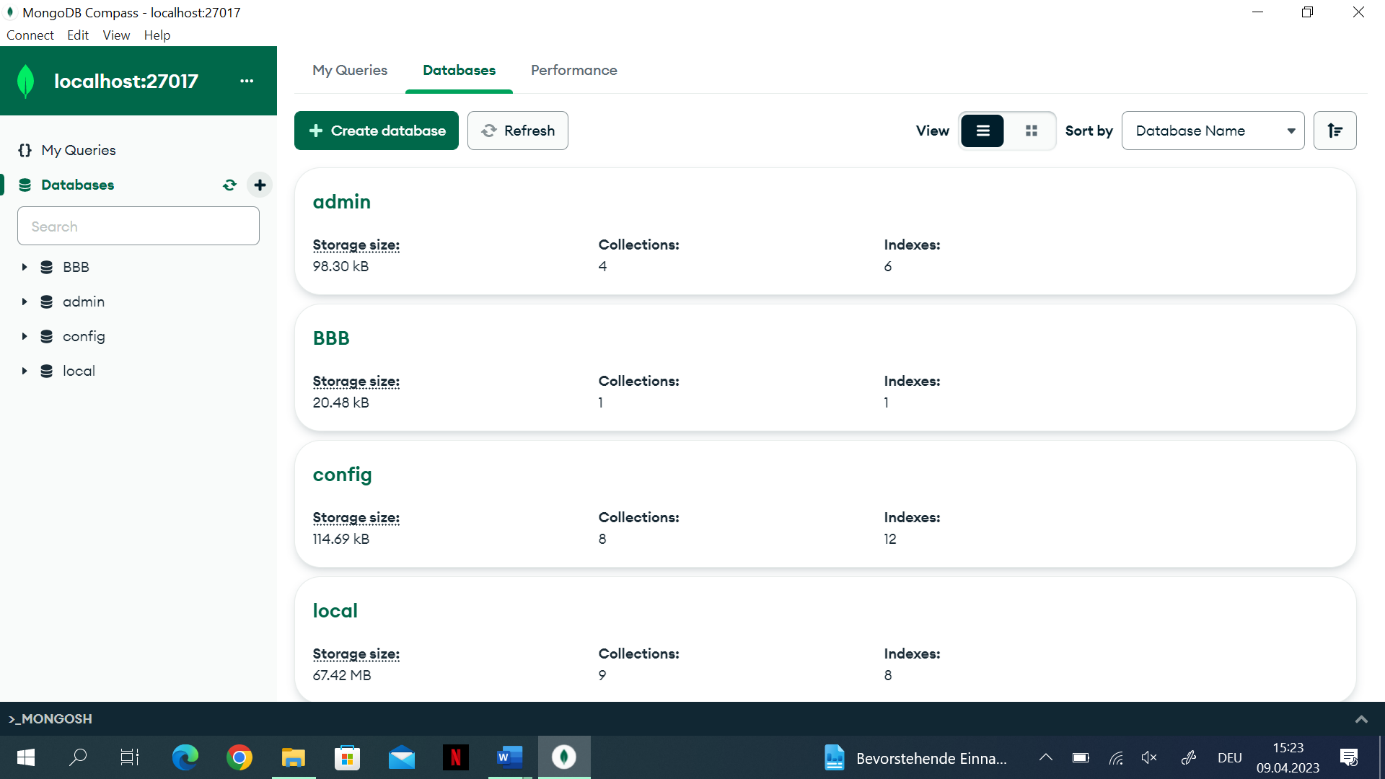
Hier habe ich alle Zahlen, die in Anführungszeichen standen und einen Doppelpunkt, ein Leerzeichen und eine geschweifte Klammer hatten mit einer geschweiften Klammer ersetzt. (Ich habe alles vor der geschweiften Klammer gelöscht)

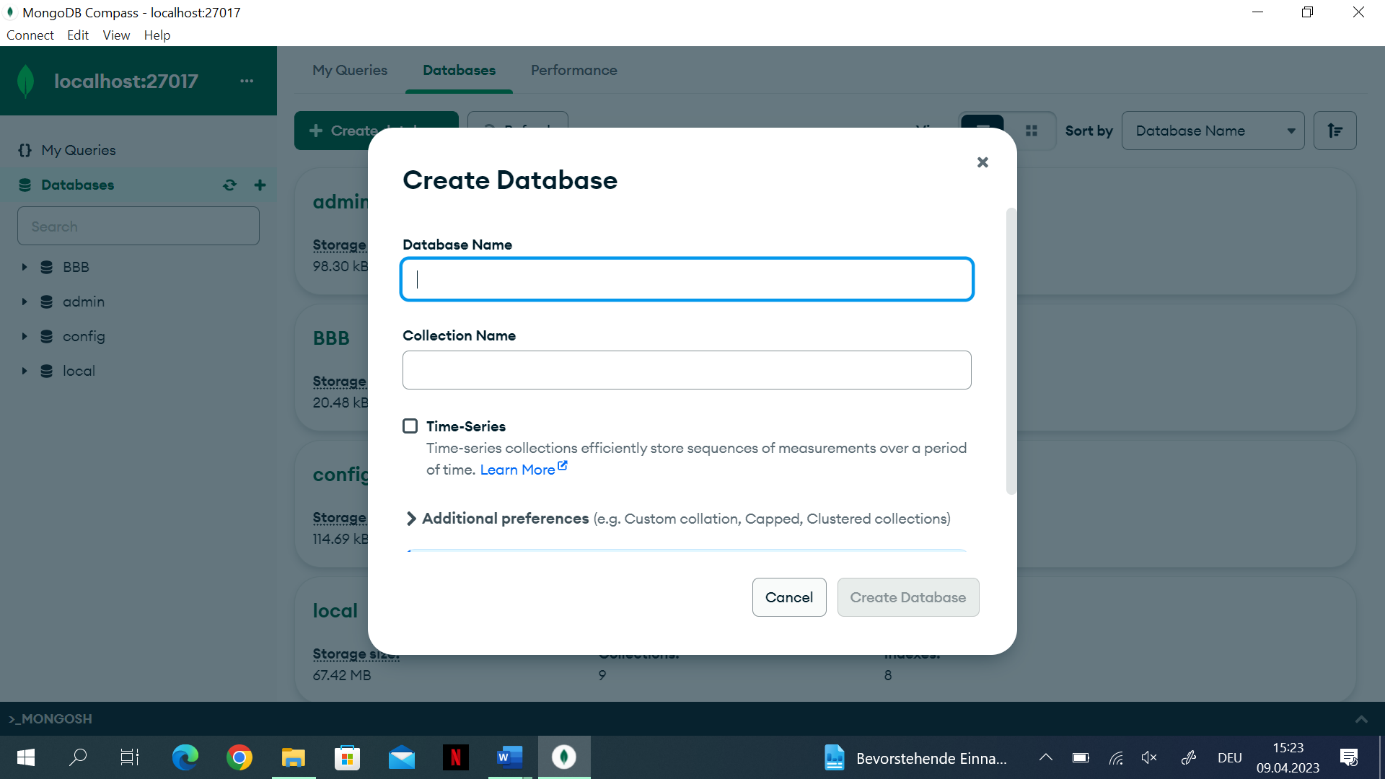
|  |  |
| --- | --- |
| Suchen | "\d+":\s{ |
| Ersetzen | { |

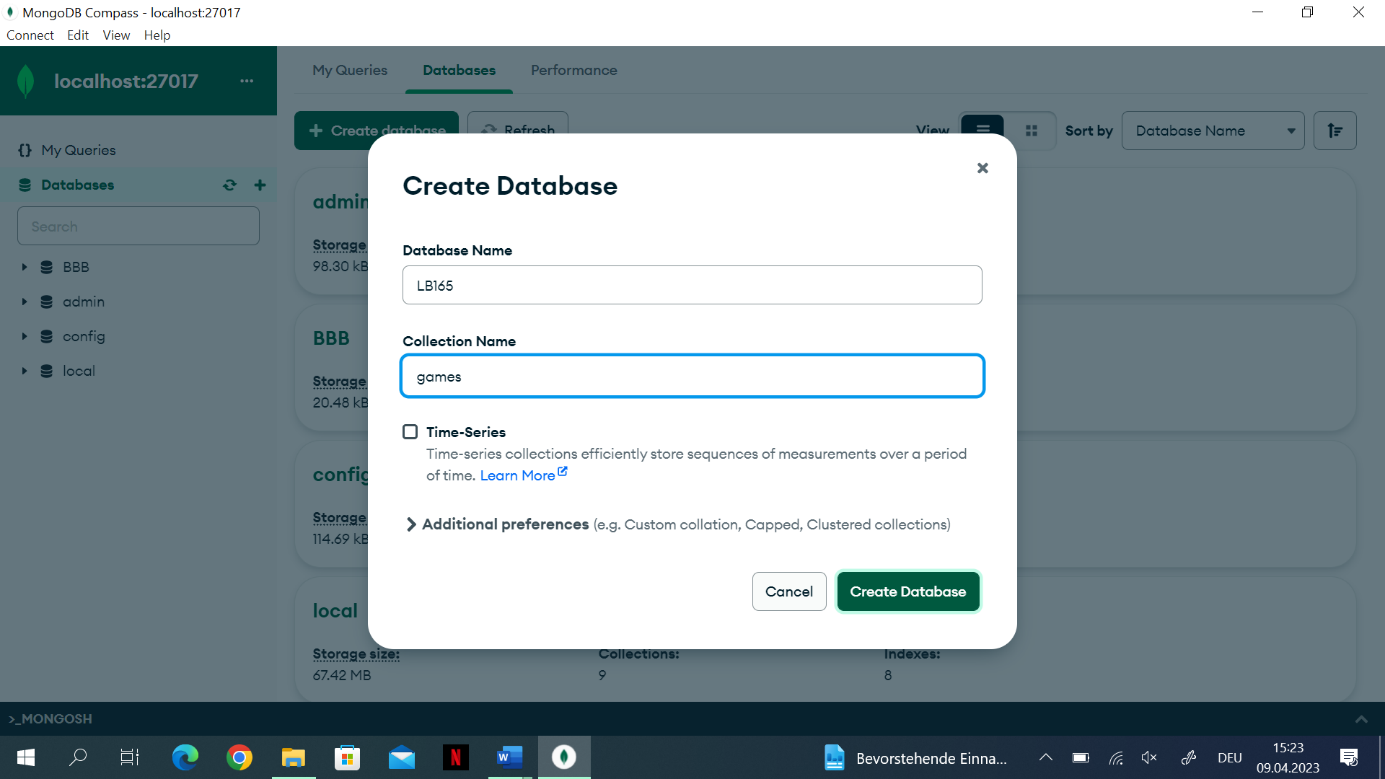
Hier habe ich bei allen Werten bei «price», «initialprice» und «discount», welche in Anführungszeichen standen, die Anführungszeichen gelöscht.

|  |  |
| --- | --- |
| Suchen | ("price":\s\*|"initialprice":\s\*|"discount":\s\*)"(.\*?)" |
| Ersetzen | \1\2 |

### Importieren

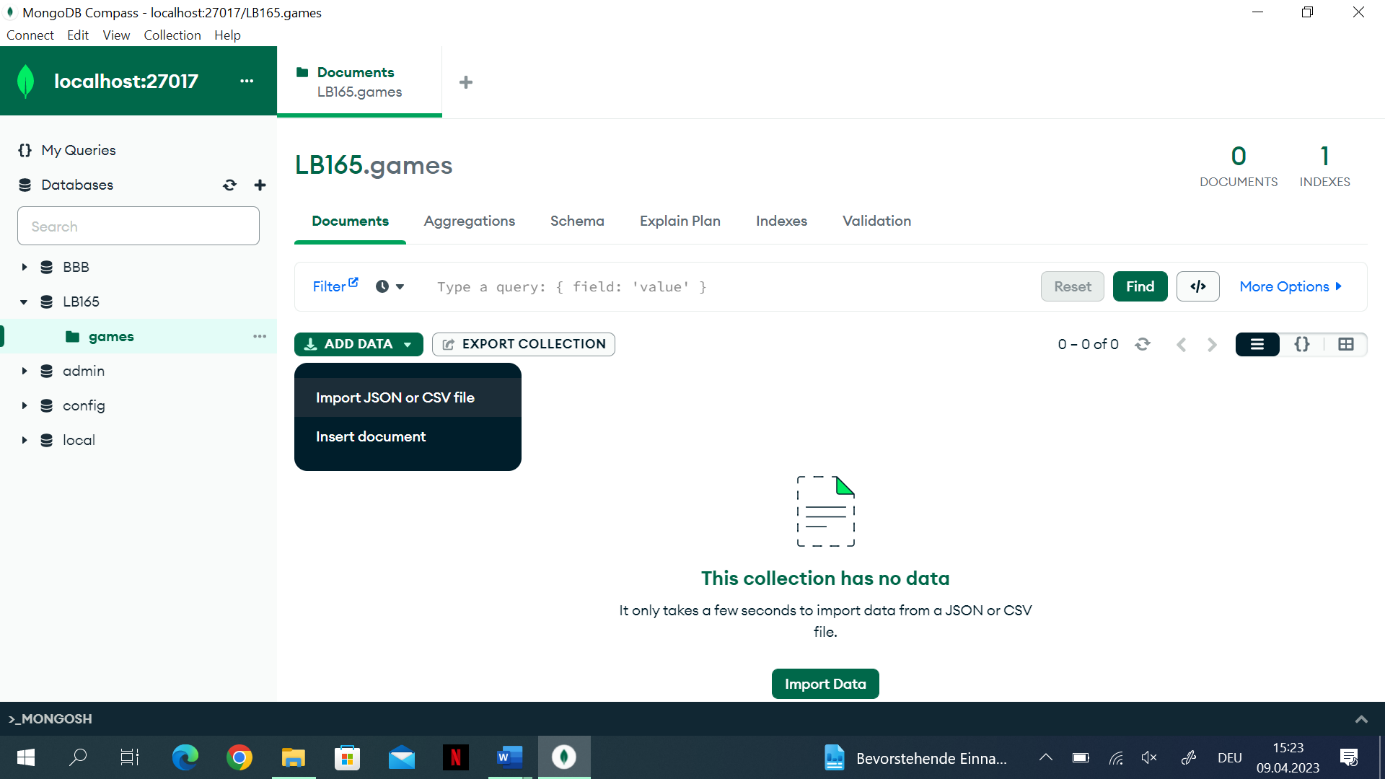


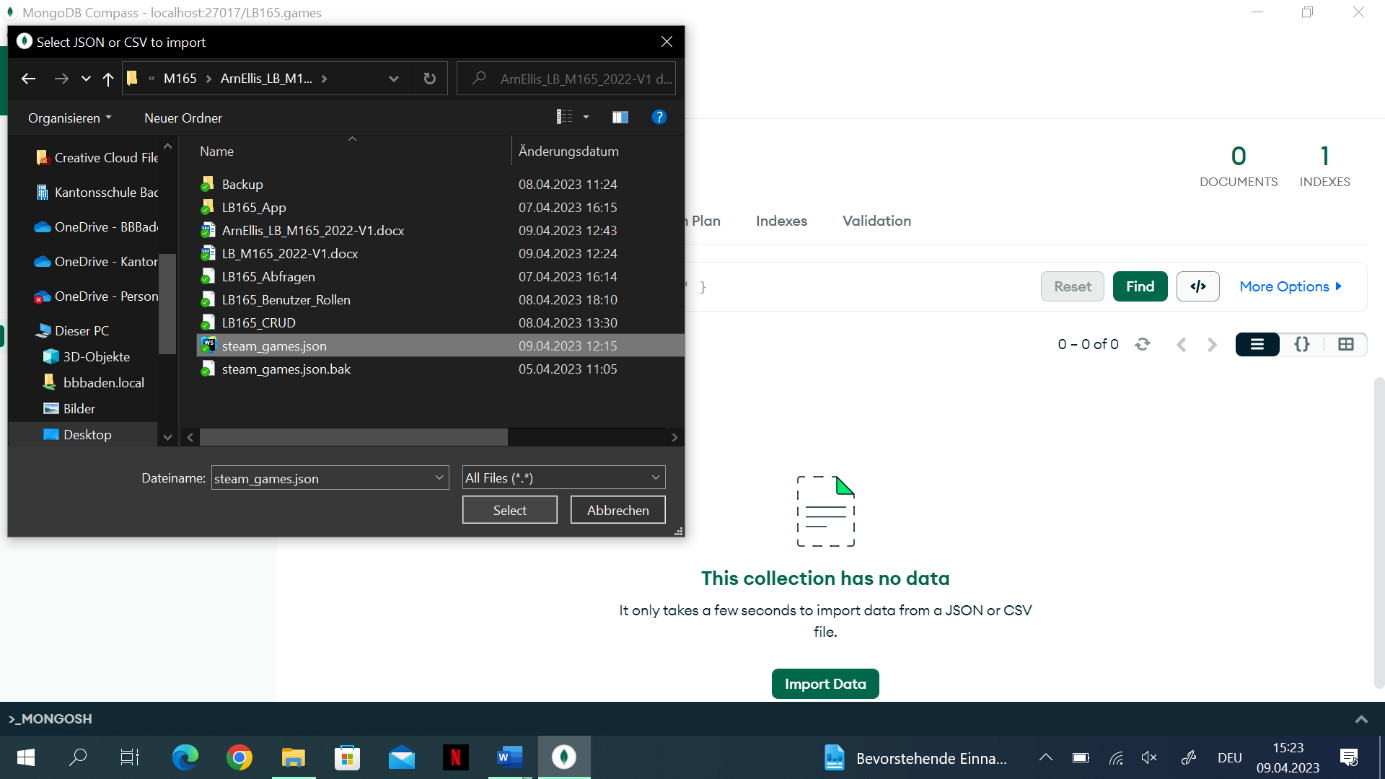


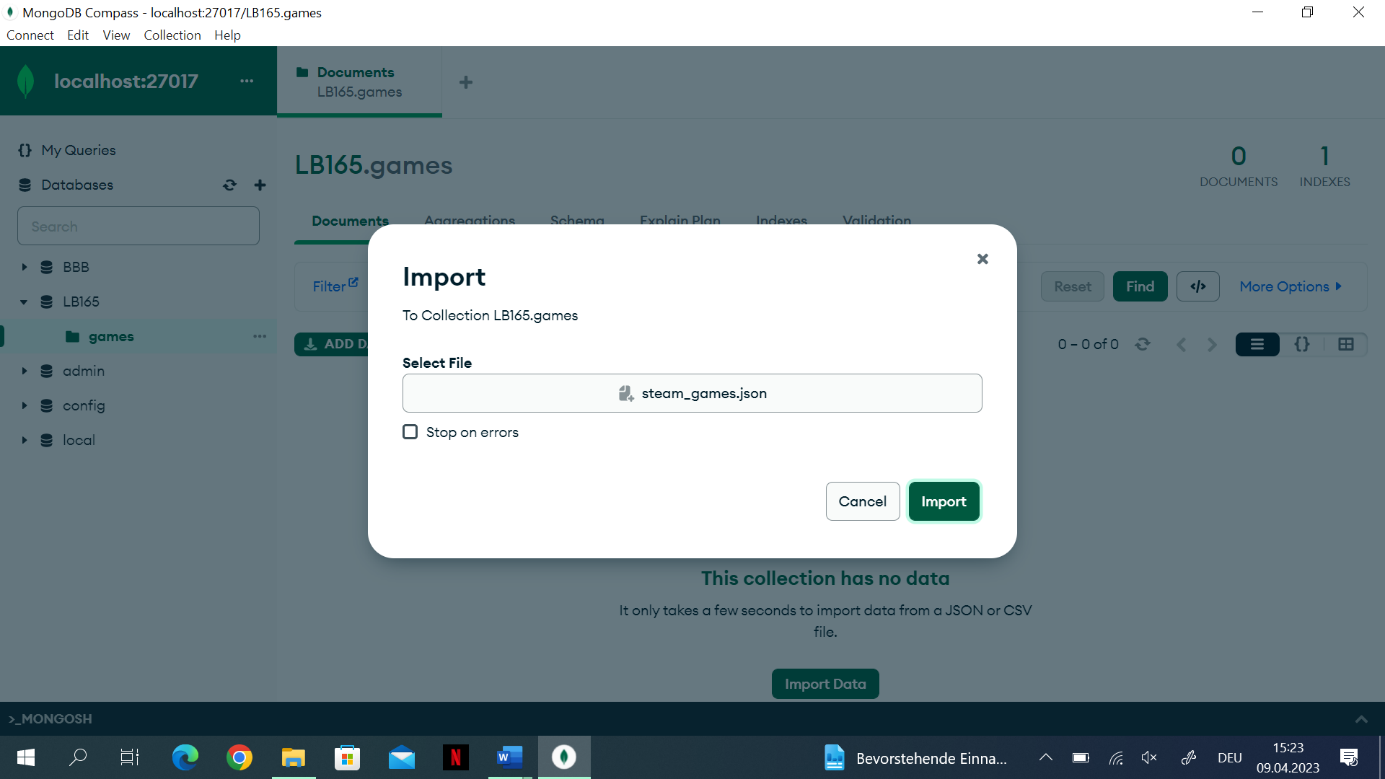


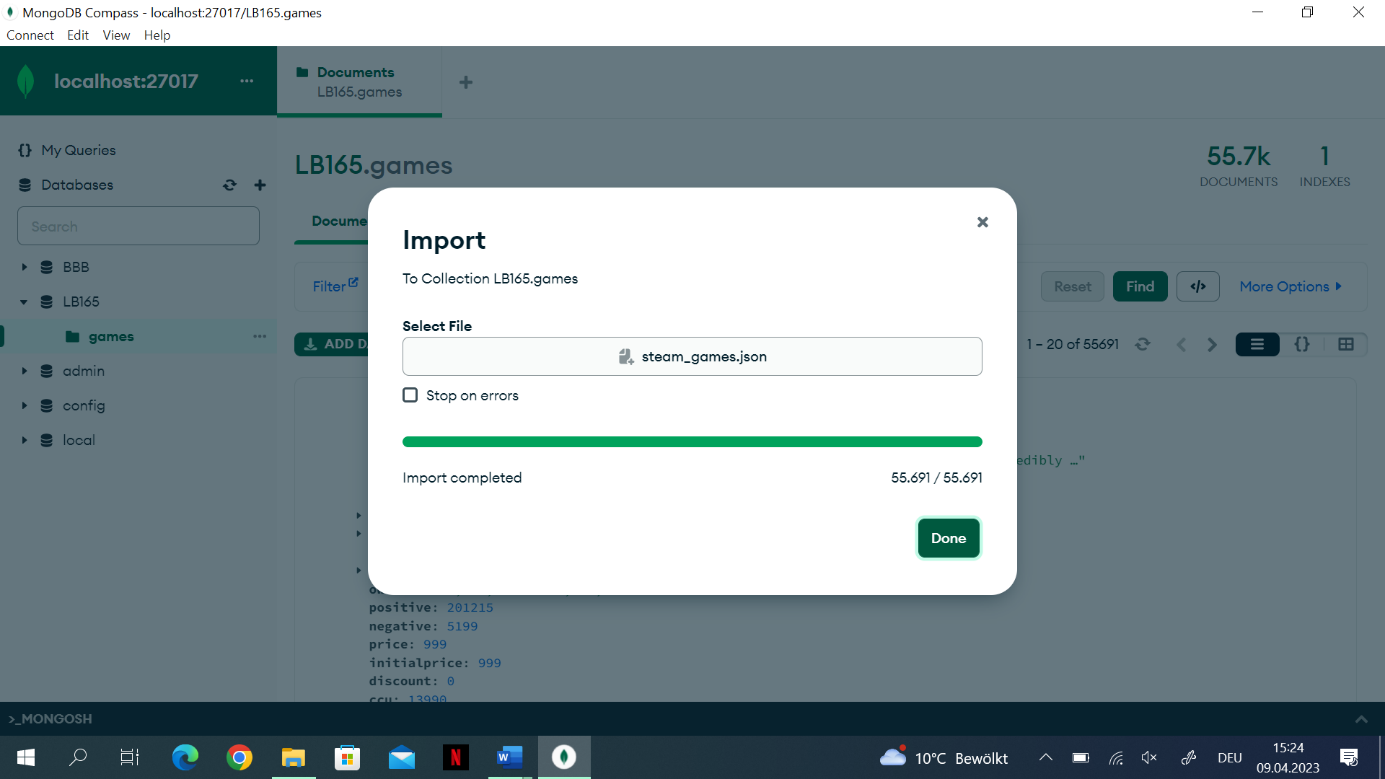
Ein Bild, das Text enthält.

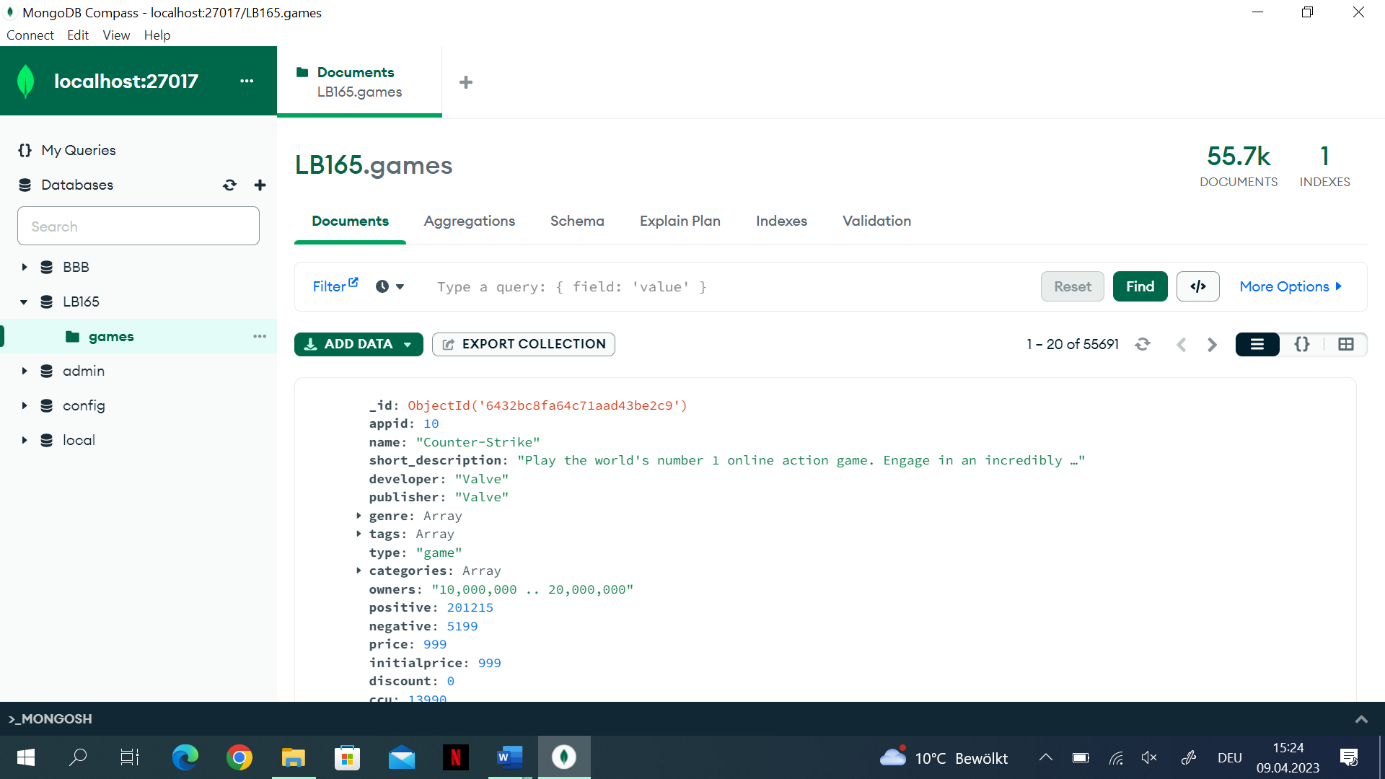
Automatisch generierte Beschreibung











## Ich kann Indices erstellen sowie Create-, Insert-, Update- und Delete-Operationen am Datenbestand einer NoSQL-Datenbank durchführen.

Index

Ich habe hier ein Index vom feld «name» gemacht, da ich dieses am häufigsten bei Abfragen mitgebe.  
db.games.createIndex({name: 1})  
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

### CRUD

// Einfügen eines einzelnen Datensatzes

db.games.insertOne({

  appid: 123,

  name: 'Example Game',

  short\_description: 'Example short description',

  developer: 'Example Developer',

  publisher: 'Example Publisher',

  genre: ['Action', 'Adventure'],

  tags: ['Action', 'Adventure', 'Indie'],

  type: 'game',

  categories: ['Single-player', 'Partial Controller Support'],

  owners: '200,000 .. 500,000',

  positive: 100,

  negative: 20,

  price: '999',

  initialprice: '999',

  discount: '0',

  ccu: 0,

  languages: ['English'],

  platforms: {

    windows: true,

    mac: false,

    linux: false,

  },

  release\_date: '2020/05/14',

  required\_age: 0,

  website: '',

  header\_image: 'https://example.com/header.jpg',

})

// Einfügen mehrerer Datensätze

db.games.insertMany([

  {

    appid: 456,

    name: 'Example Game 2',

    short\_description: 'Example short description 2',

    developer: 'Example Developer',

    publisher: 'Example Publisher',

    genre: ['Strategy', 'Simulation'],

    tags: ['Strategy', 'Simulation', 'Indie'],

    type: 'game',

    categories: ['Single-player', 'Full Controller Support'],

    owners: '100,000 .. 200,000',

    positive: 200,

    negative: 40,

    price: '1999',

    initialprice: '1999',

    discount: '0',

    ccu: 0,

    languages: ['English', 'German'],

    platforms: {

      windows: true,

      mac: true,

      linux: false,

    },

    release\_date: '2020/06/10',

    required\_age: 0,

    website: '',

    header\_image: 'https://example.com/header2.jpg',

  },

  {

    appid: 789,

    name: 'Example Game 3',

    short\_description: 'Example short description 3',

    developer: 'Example Developer 2',

    publisher: 'Example Publisher 2',

    genre: ['RPG', 'Adventure'],

    tags: ['RPG', 'Adventure', 'Indie'],

    type: 'game',

    categories: ['Single-player', 'Steam Achievements'],

    owners: '50,000 .. 100,000',

    positive: 300,

    negative: 60,

    price: '2999',

    initialprice: '2999',

    discount: '0',

    ccu: 0,

    languages: ['English', 'French'],

    platforms: {

      windows: true,

      mac: true,

      linux: true,

    },

    release\_date: '2020/07/05',

    required\_age: 0,

    website: '',

    header\_image: 'https://example.com/header3.jpg',

  },

])

// Aktualisieren eines einzelnen Datensatzes

db.games.updateOne({ appid: 123 }, { $set: { price: 799 } })

// Aktualisieren mehrerer Datensätze

db.games.updateMany(

  { developer: 'Example Developer' },

  { $set: { publisher: 'New Publisher' } }

)

// Löschen eines einzelnen Datensatzes

db.games.deleteOne({ appid: 789 })

// Löschen mehrerer Datensätze

db.games.deleteMany({ developer: 'Example Developer' })

# Handlungsziel 3

## Ich definiere und begründe in Bezug auf den Anwendungsfall passende Zugriffsberechtigungen (Benutzende, Rollen) und implementiere diese in einem NoSQL-DBMS.

Ich habe zwei Rollen erstellt und diese zwei Benutzern zugeteilt.  
Die Rollen:

1. **gameViewer**-Rolle: Diese Rolle ist für Benutzer gedacht, die Spiele durchsuchen, Details anzeigen und nach Tags, Genres, Entwicklern usw. filtern möchten, ohne die Möglichkeit zu haben, die Daten zu ändern.
2. **gameManager**-Rolle: Diese Rolle ist für Benutzer gedacht, die für die Verwaltung der Spiele-Datenbank verantwortlich sind. Sie können Spiele hinzufügen, aktualisieren, löschen und alle Aktionen ausführen, die der **gameViewer** ausführen kann.

Die **gameViewer**-Rolle habe ich einem Benutzer gegeben, den ich **viewerUser** genannt habe. Die **gameManager**-Rolle habe ich dem **managerUser** gegeben.

### Rollen

// Rollen erstellen

use LB165

// Rolle gameViewer

db.createRole({

  role: 'gameViewer',

  privileges: [

    { resource: { db: 'LB165', collection: 'games' }, actions: ['find'] },

  ],

  roles: [],

})

// Rolle gameManager

db.createRole({

  role: 'gameManager',

  privileges: [

    {

      resource: { db: 'LB165', collection: 'games' },

      actions: ['find', 'insert', 'update', 'remove'],

    },

  ],

  roles: [],

})

### Benutzer

// Benutzer erstellen

use LB165

// Benutzer mit der Rolle gameViewer

db.createUser({

  user: 'viewerUser',

  pwd: 'viewerPassword',

  roles: [{ role: 'gameViewer', db: 'LB165' }],

})

// Benutzer mit der Rolle gameManager

db.createUser({

  user: 'managerUser',

  pwd: 'managerPassword',

  roles: [{ role: 'gameManager', db: 'LB165' }],

})

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte Beschreibung

# Handlungsziel 4

## Ich kann zwei Gründe für die Durchführung von Backups benennen.

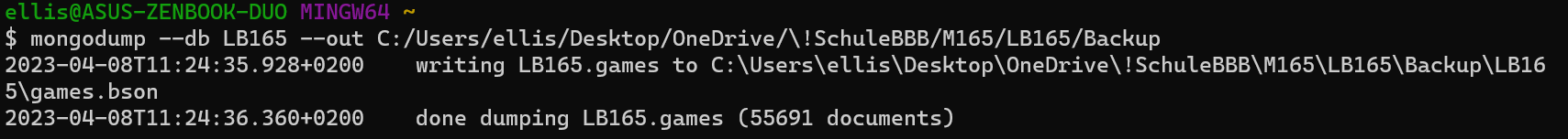
1. Datenverlust verhindern: Ein wichtiger Grund für die Durchführung von Backups ist der Schutz vor Datenverlust. Durch regelmäßige Backups wird sichergestellt, dass kritische Daten wiederhergestellt werden können, wenn ein Datenverlust auftritt.
2. Verbesserung der Produktivität: Datenverlust kann zu Verzögerungen und Unterbrechungen im Arbeitsablauf führen, da Mitarbeiter möglicherweise auf wichtige Informationen oder Ressourcen nicht zugreifen können. Mit regelmäßigen Backups können Unternehmen sicherstellen, dass Mitarbeiter schnell wieder auf ihre Daten zugreifen können, wodurch die Produktivität erhalten bleibt.

## Ich kann ein Backup-Verfahren für eine NoSQL-DBMS implementieren und die Funktionsweise mit der Anwendung eines Wiederherstellungsverfahren belegen.

### Backup

mongodump --db LB165 --out C:/Users/ellis/Desktop/OneDrive/\!SchuleBBB/M165/LB165/Backup

Ich habe /\!SchuleBBB geschrieben, weil ich das Ausrufezeichen mit dem Backslash «auskommentieren» musste.



### Löschen

use LB165  
db.dropDatabase()

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

### Restore

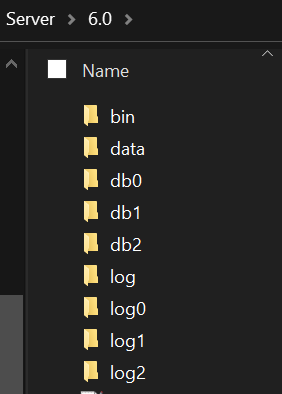
mongorestore --db LB165 –dir C:/Users/ellis/Desktop/OneDrive/\!SchuleBBB/M165/LB165/Backup/LB165Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

# Handlungsziel 5

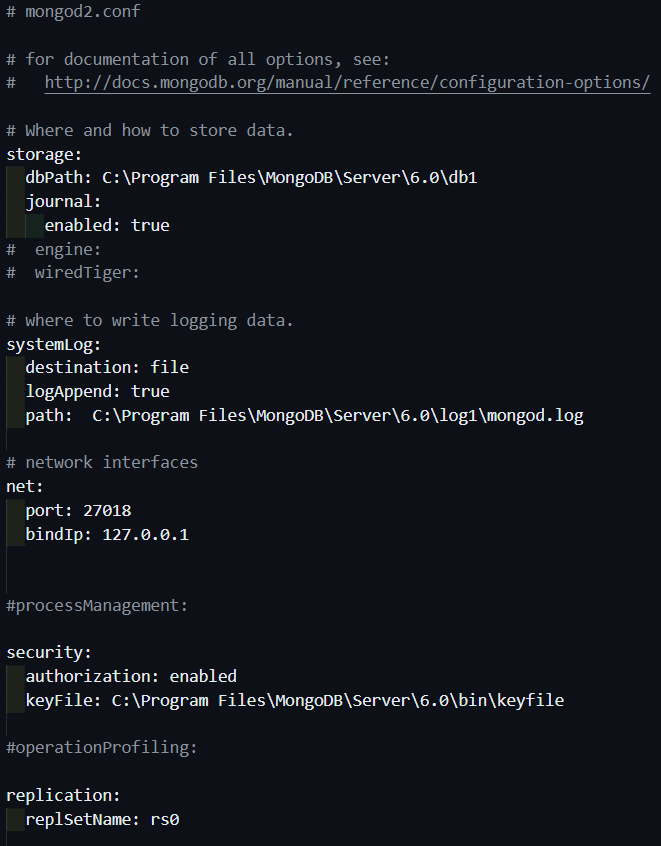
## Ich kann für einen Anwendungsfall eine geeignete Replikationsart auswählen und erläutern, die Vorteile dieser Lösung im Vergleich zu Varianten aufzeigen und diese in Betrieb nehmen.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung 

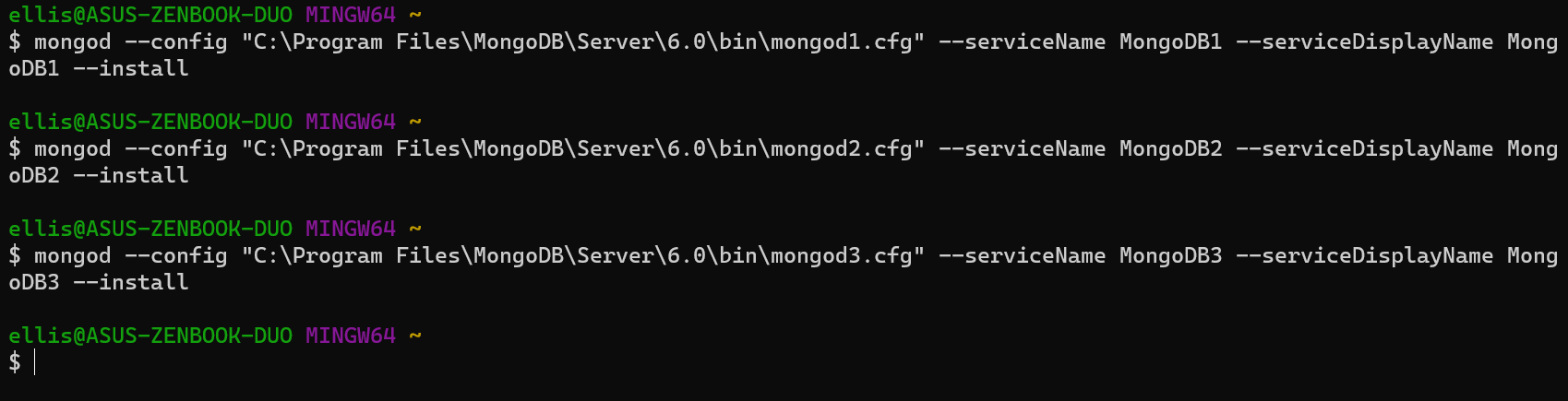
Ein Bild, das Text enthält.

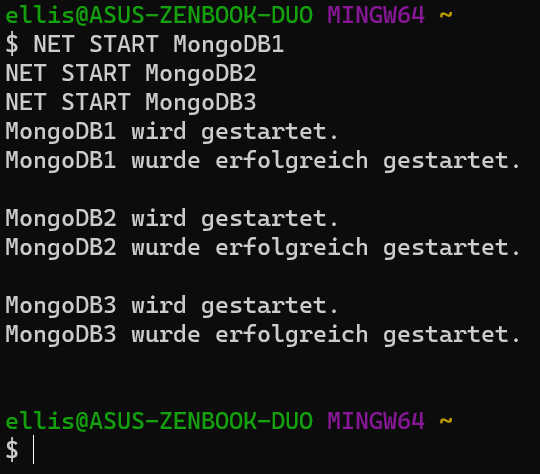
Automatisch generierte Beschreibung



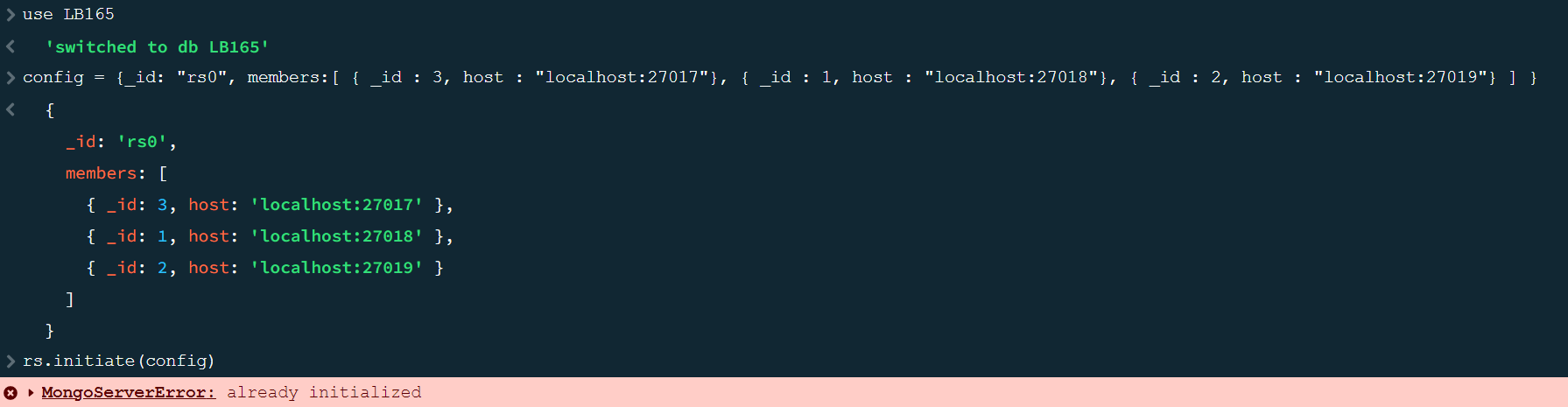
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung





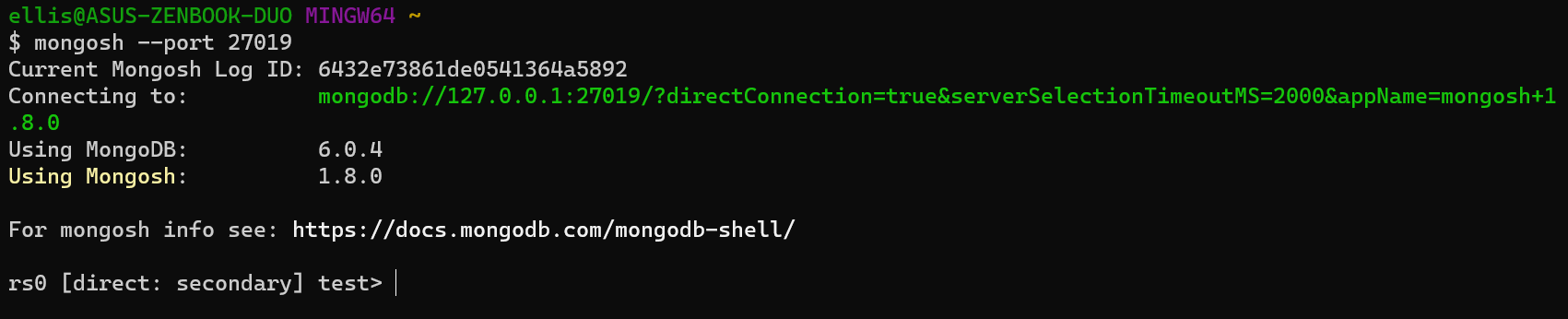


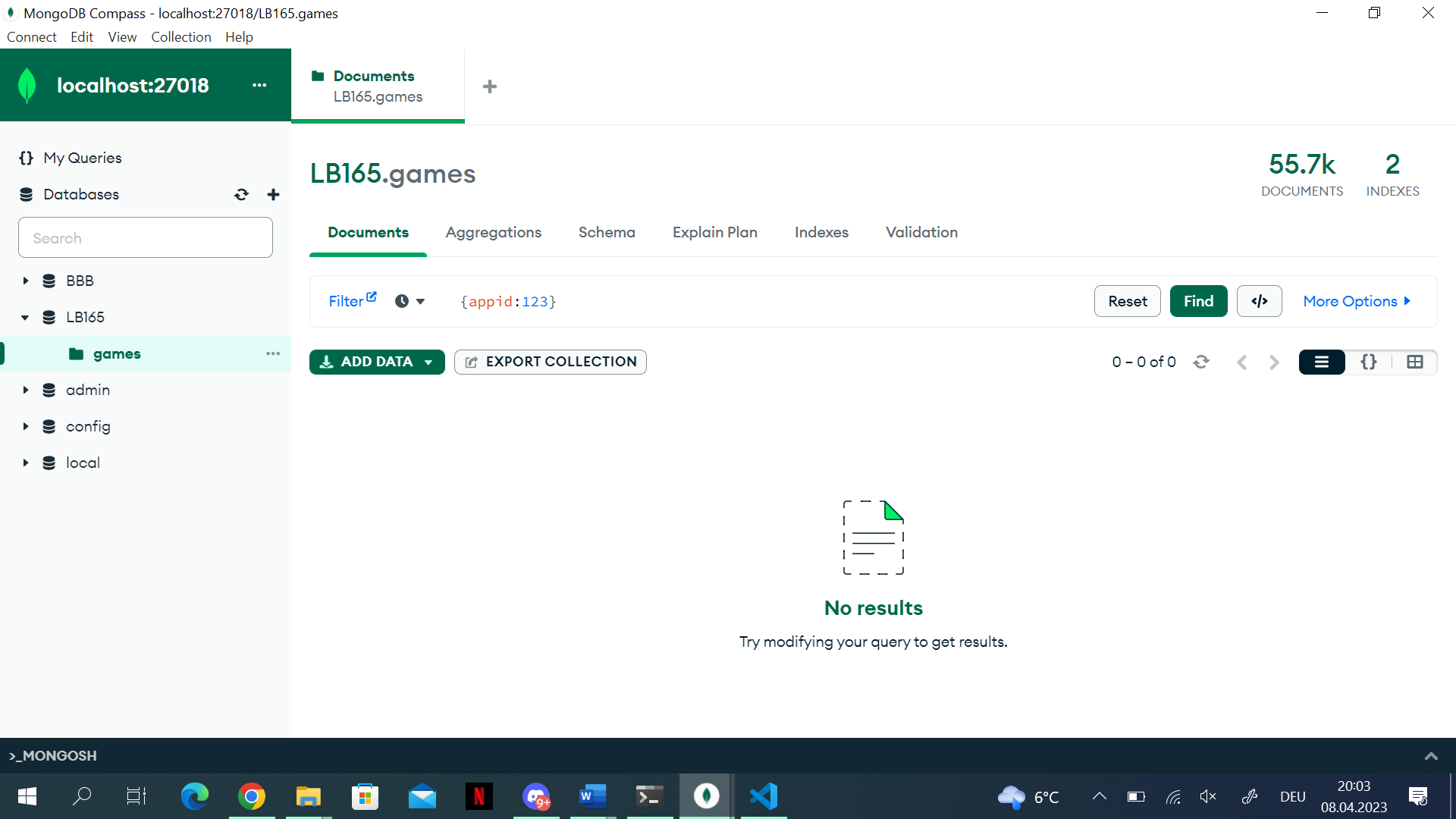


Bei mir kommt hier ein Fehler, weil ich es bereits gestartet, aber nicht dokumentiert habe.



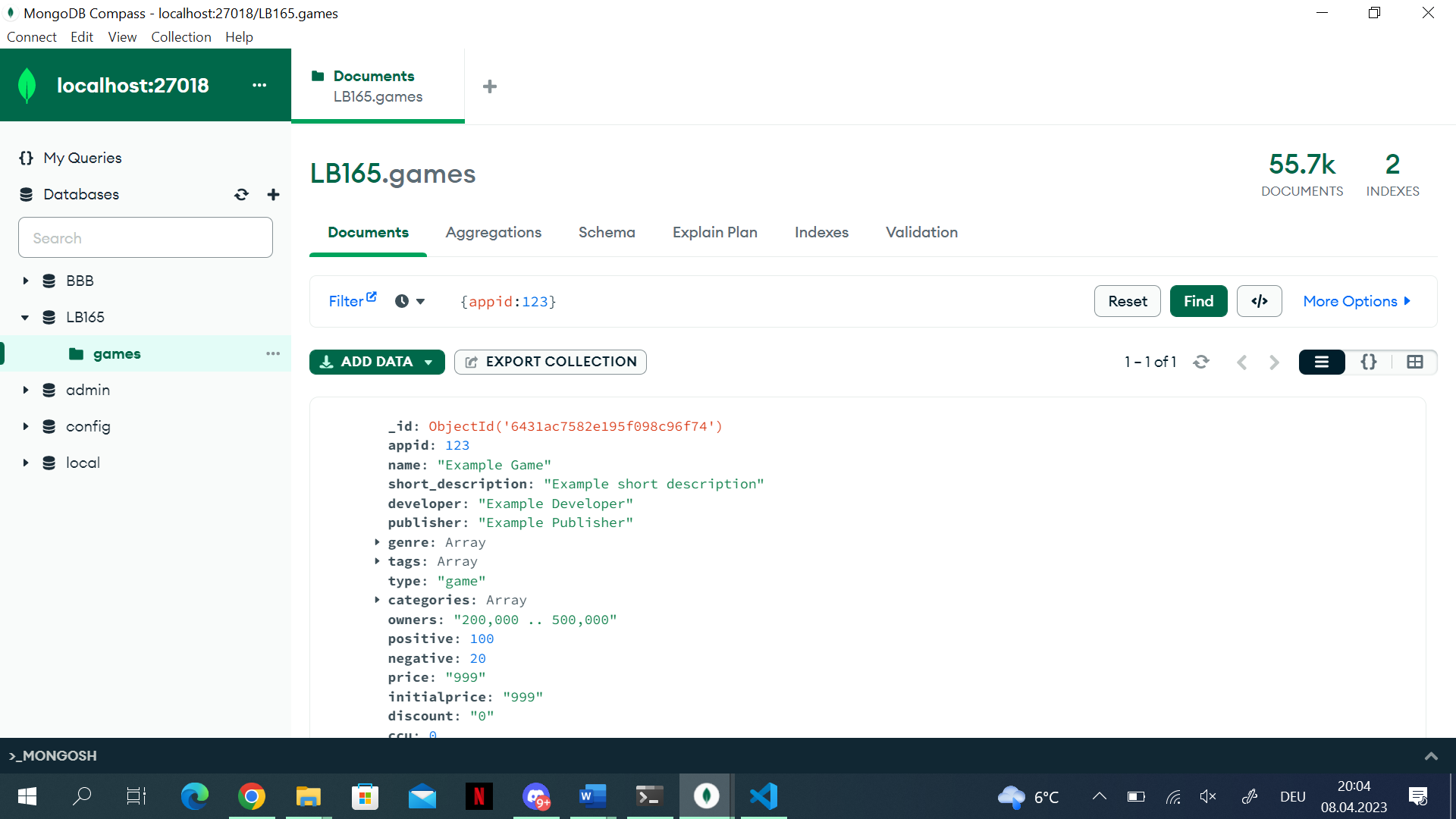






Ein Bild, das Text enthält.

Automatisch generierte Beschreibung



# Handlungsziel 6

## Ich kann Abfragen mit verschiedener Struktur (Selektion, Projektion) anwenden und dabei unterschiedliche Verfahren (Aggregation, parallele Verarbeitung) nutzen.

### Selektion und Projektion

// Finde alle Spiele, die von "Valve" entwickelt wurden und gib nur den Namen und das Erscheinungsdatum zurück

db.games.find({ developer: 'Valve' }, { name: 1, release\_date: 1 })

// Finde alle Spiele, die im Jahr 2020 veröffentlicht wurden und gib nur den Namen und das Genre zurück

db.games.find({ release\_date: { $regex: '^2020' } }, { name: 1, genre: 1 })

// Finde alle Spiele mit dem Tag "Multiplayer" und gib nur den Namen und die Plattformen zurück

db.games.find({ tags: 'Multiplayer' }, { name: 1, platforms: 1 })

// Finde alle Spiele, die auf Windows und Linux verfügbar sind, und gib nur den Namen und den Preis zurück

db.games.find(

  { 'platforms.windows': true, 'platforms.linux': true },

  { name: 1, price: 1 }

)

// Finde alle Spiele, die von "SEGA" entwickelt wurden und mehr als 100 positive Bewertungen haben, und gib nur den Namen und die Anzahl der positiven Bewertungen zurück

db.games.find(

  { developer: 'SEGA', positive: { $gt: 100 } },

  { name: 1, positive: 1 }

)

// Finde alle Spiele, die mindestens 50 % Rabatt haben, und gib nur den Namen, den ursprünglichen Preis und den Rabatt zurück

db.games.find(

  { discount: { $gte: '50' } },

  { name: 1, initialprice: 1, discount: 1 }

)

### Aggregationen

// Finde für jeden «Tag» den durchschnittlichen Preis und gib den Namen des Tags und den Durchschnittspreis zurück

db.games.aggregate([

  { $match: { price: { $ne: '0' }, tags: { $ne: [] } } },

  { $unwind: '$tags' },

  {

    $group: {

      \_id: '$tags',

      averagePrice: { $avg: { $toDouble: '$price' } },

    },

  },

  {

    $project: {

      tag: '$\_id',

      averagePrice: {

        $divide: [

          {

            $multiply: [{ $round: [{ $divide: ['$averagePrice', 5] }] }, 5],

          },

          100,

        ],

      },

      \_id: 0,

    },

  },

  { $sort: { tag: 1 } },

])

// Finde die durchschnittlichen positiven Bewertungen für Spiele, die im Jahr 2021 veröffentlicht wurden

db.games.aggregate([

  { $match: { release\_date: { $regex: '^2021' } } },

  { $group: { \_id: null, avg\_positive: { $avg: '$positive' } } },

])

// Zähle die Anzahl der Spiele für jedes Genre

db.games.aggregate([

  { $unwind: '$genre' },

  { $group: { \_id: '$genre', count: { $sum: 1 } } },

])

// Finde die 5 am häufigsten verwendeten Tags in den Spielen

db.games.aggregate([

  { $unwind: '$tags' },

  { $group: { \_id: '$tags', count: { $sum: 1 } } },

  { $sort: { count: -1 } },

  { $limit: 5 },

])

// Berechne den durchschnittlichen Preis für Spiele, die von "Valve" entwickelt wurden

db.games.aggregate([

  { $match: { developer: 'Valve' } },

  { $group: { \_id: null, avg\_price: { $avg: '$price' } } },

])

// Finde die Anzahl der Spiele für jede Plattform (Windows, Mac, Linux)

db.games.aggregate([

  {

    $group: {

      \_id: null,

      windows: {

        $sum: { $cond: [{ $eq: ['$platforms.windows', true] }, 1, 0] },

      },

      mac: { $sum: { $cond: [{ $eq: ['$platforms.mac', true] }, 1, 0] } },

      linux: { $sum: { $cond: [{ $eq: ['$platforms.linux', true] }, 1, 0] } },

    },

  },

])

## Ich kann eine Verbindung zwischen einer Programmiersprache und einem NoSQL-DBMS herstellen und CRUD-Operationen ausführen

using MongoDB.Bson;  
using MongoDB.Driver;  
  
namespace LB165\_App;  
  
internal class Program  
{  
 static void Main(string[] args)  
 {  
 *// Verbindung zur MongoDB-Instanz herstellen* MongoClient client = new MongoClient("mongodb://localhost:27017");  
 IMongoDatabase database = client.GetDatabase("LB165");  
 IMongoCollection<BsonDocument> games = database.GetCollection<BsonDocument>("games");  
  
 *// Create-Operation: Ein neues Spiel einfügen* var data = new BsonDocument {  
 {"appid", 12345},  
 {"name", "Testspiel"},  
 {"positive", 100},  
 {"negative", 10}  
 };  
 games.InsertOne(data);  
 Console.WriteLine("--------------------------------------------");  
 Console.WriteLine("Ein neues Spiel wurde hinzugefügt. (appid: 12345)");  
 Console.WriteLine("--------------------------------------------");  
   
 *// Read-Operation: Spiele mit mehr als 100.000 positiven Bewertungen finden* var filter = Builders<BsonDocument>.Filter.Gt("positive", 100000);  
 var result = games.Find(filter).ToList();  
 Console.WriteLine($"Spiele mit mehr als 100.000 positiven Bewertungen: {result.Count}");  
 Console.WriteLine("--------------------------------------------");

*// Update-Operation: Die Anzahl der positiven Bewertungen für ein Spiel erhöhen* var updateFilter = Builders<BsonDocument>.Filter.Eq("appid", 12345);  
 var update = Builders<BsonDocument>.Update.Inc("positive", 11);  
 var projection = Builders<BsonDocument>.Projection.Include("name").Include("appid").Include("positive").Exclude("\_id");   
 var foundFirstDocument = games.Find(updateFilter).FirstOrDefault();  
   
 if (foundFirstDocument != null)  
 {  
 Console.WriteLine("Name: " + foundFirstDocument["name"]);  
 Console.WriteLine("AppID: " + foundFirstDocument["appid"]);  
 Console.WriteLine("Positive Bewertungen: " + foundFirstDocument["positive"]);  
 Console.WriteLine("--------------------------------------------");  
 }  
 else  
 {  
 Console.WriteLine("Kein Dokument mit der angegebenen AppID gefunden.");  
 }  
 games.UpdateOne(updateFilter, update);  
 Console.WriteLine("Beim Dokument mit der appid 12345 wurden die positiven Bewertungen um 11 erhöht");  
 Console.WriteLine("--------------------------------------------");  
 var foundSecondDocument = games.Find(updateFilter).Project(projection).FirstOrDefault();  
 if (foundSecondDocument != null)  
 {  
 Console.WriteLine("Name: " + foundSecondDocument["name"]);  
 Console.WriteLine("AppID: " + foundSecondDocument["appid"]);  
 Console.WriteLine("Positive Bewertungen: " + foundSecondDocument["positive"]);  
 Console.WriteLine("--------------------------------------------");  
 }  
 else  
 {  
 Console.WriteLine("Kein Dokument mit der angegebenen AppID gefunden.");  
 Console.WriteLine("--------------------------------------------");  
 }  
  
 *// Delete-Operation: Ein Spiel löschen* var deleteFilter = Builders<BsonDocument>.Filter.Lt("appid", 12345);  
 games.DeleteOne(deleteFilter);  
 Console.WriteLine("Das Spiel mit der AppID 12345 wurde gelöscht.");  
 Console.WriteLine("--------------------------------------------");  
 }  
}