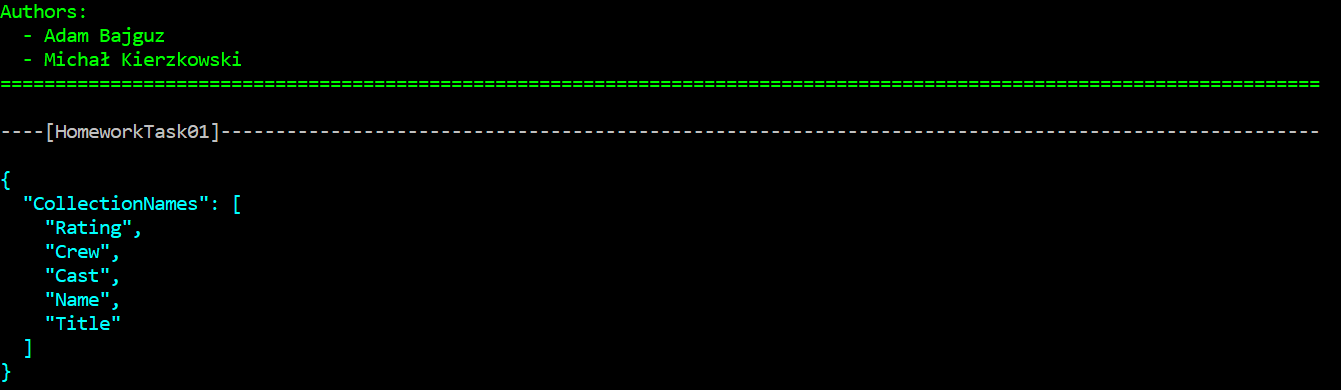
|  |  |  |
| --- | --- | --- |
| Wydział Informatyki Politechniki Białostockiej  Zaawansowane bazy danych i hurtownie danych  Pracownia specjalistyczna | | Data realizacji:  24.05.2020r. |
| Zadanie nr 5 | | Prowadzący:  dr hab. inż. Agnieszka Drużdżel, prof. PB |
| Temat: MongoDB | |
| Grupa PS 3 | Imię i nazwisko |
|  | 1. Adam Bajguz | Ocena: |
| 2. Michał Kierzkowski |

# Realizacja zadania 1

Do realizacji zadania nr 1 wykorzystano następujące polecenie „mongoimport --type tsv --db IMDB --collection nazwa\_kolekcji --headerline --file ścieżka\_do\_pliku”, które pozwoliło na załadownie wszystkich wymaganych plików jako kolekcje do bazy MongoDB. Poprawność załadownia danych zaprezentowana jest na Rys. 1.

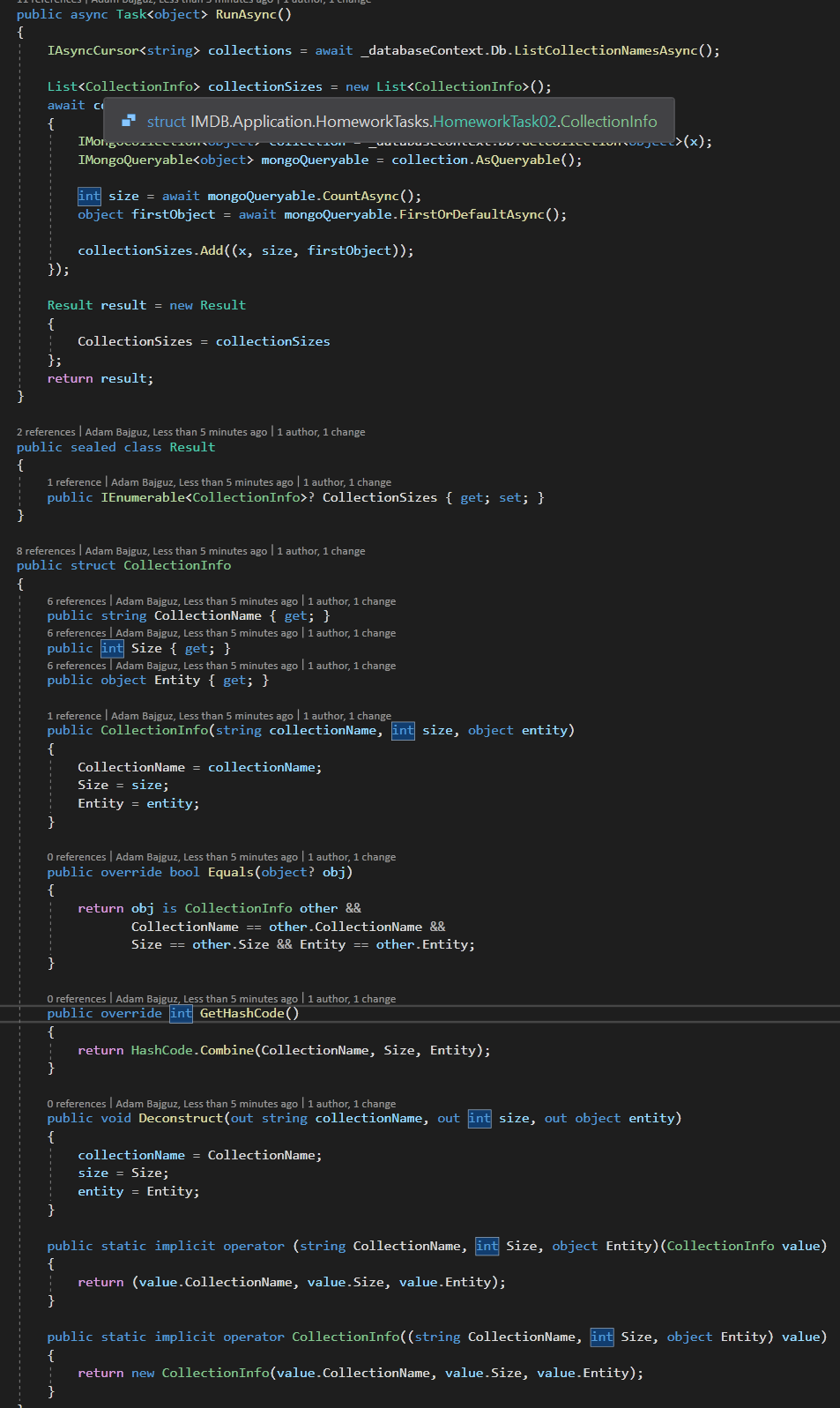




**Rys. 1.** Prezentacja poprawności załadowania danych wraz z kodem odpowiedzialnym za wypisanie kolekcji

# Realizacja zadania 1

W celu realizacji zadania nr 2 utworzono kod przedstawiony na Rys. 2.1. Wynik działania tego kodu zaprezentowano na Rys. 2.2.



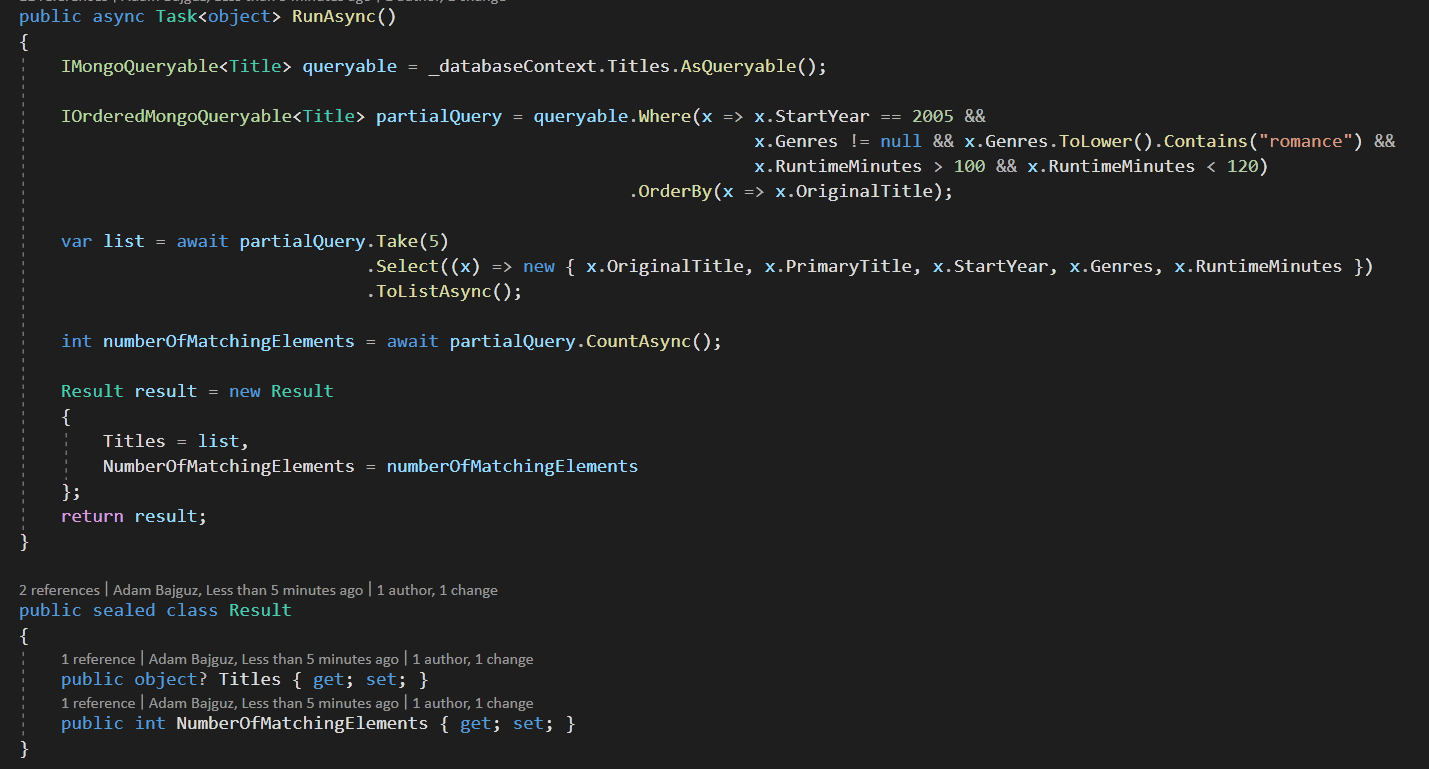
**Rys. 2.1.** Fragment najistotniejszego kodu realizującego zadanie nr 2



**Rys. 2.2.** Wynik działania kodu z Rys. 2.1

# Realizacja zadania 3

W celu realizacji zadania nr 3 utworzono kod przedstawiony na Rys. 3.1. Wynik działania tego kodu zaprezentowano na Rys. 3.2. Podczas realizacji zadania, ze względu na występowanie w bazie originalTitle i primaryTitle wybrano originalTile jako tytuł po którym sortowano wyniki.



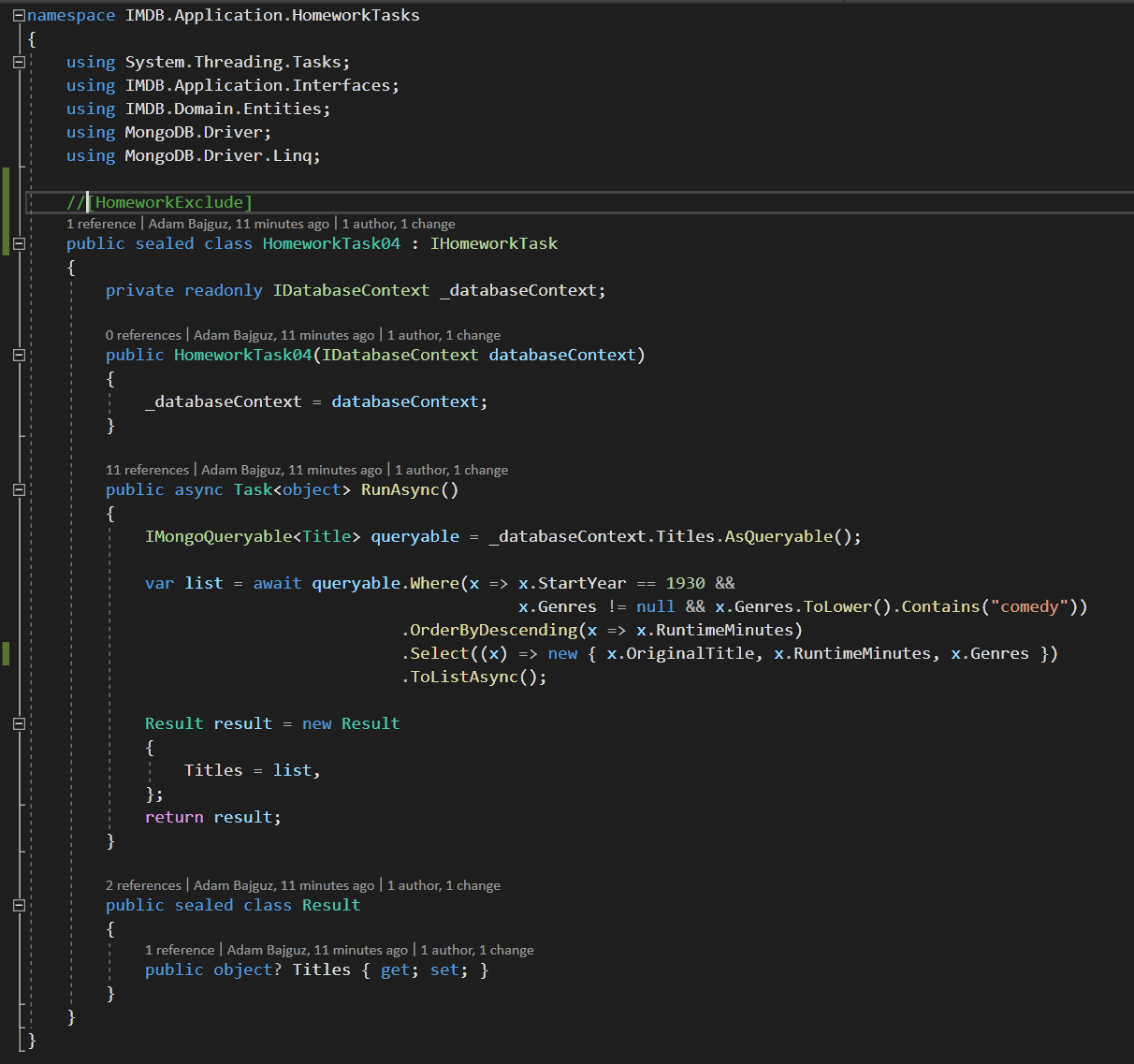
**Rys. 3.1.** Fragment najistotniejszego kodu realizującego zadanie nr 3



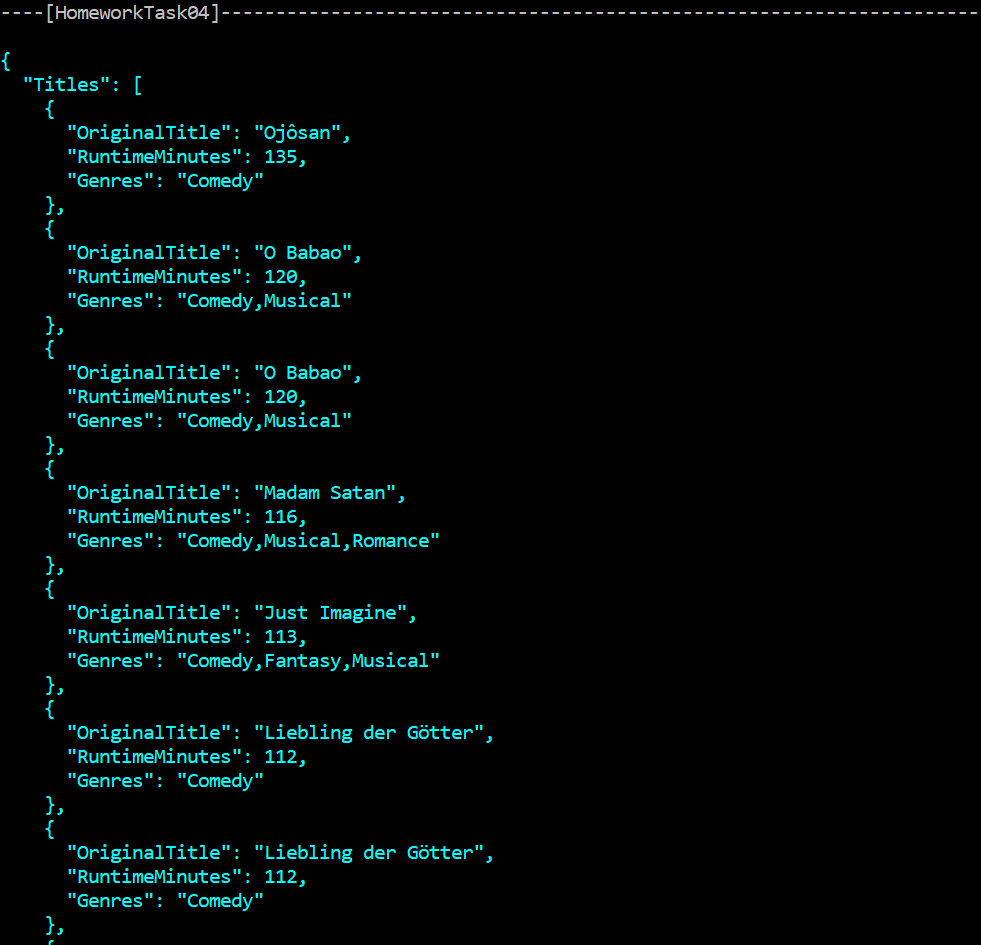
**Rys. 3.2.** Wynik działania kodu z Rys. 3.1

# Realizacja zadania 4

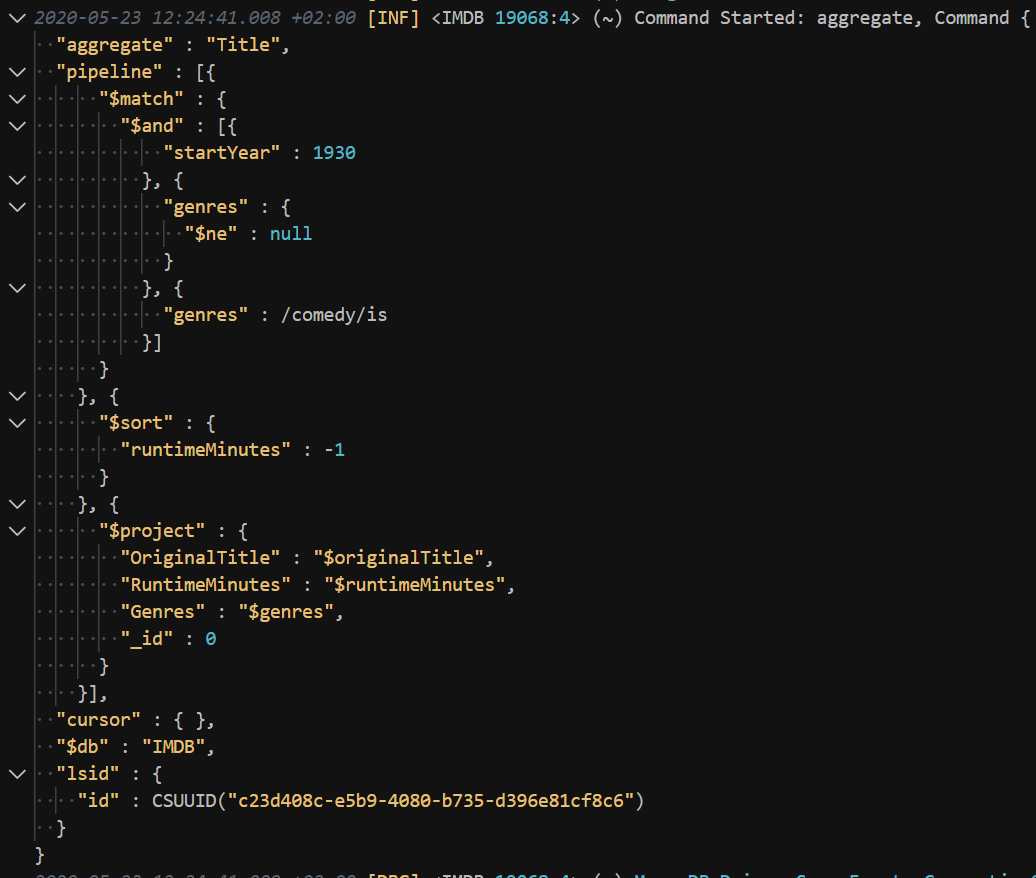
W celu realizacji zadania nr 4 utworzono kod przedstawiony na Rys. 4.1. Wynik działania tego kodu zaprezentowano na Rys. 4.2.



**Rys. 4.1.** Fragment najistotniejszego kodu realizującego zadanie nr 4



**Rys. 4.2.** Fragment wyników działania kodu z Rys. 4.1

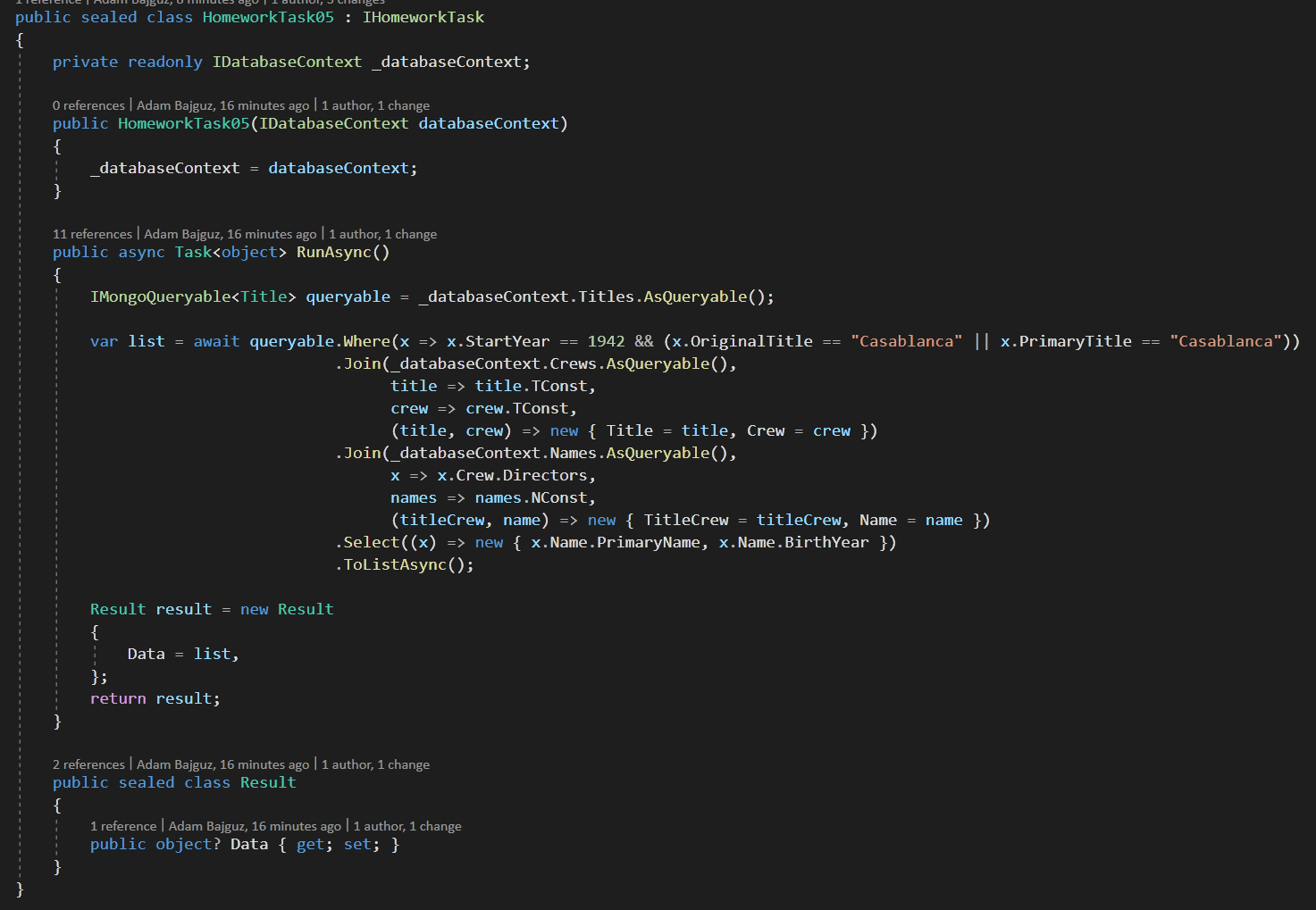


**Rys. 4.2.** Wpis z logów pokazujący rzeczywiste zapytanie zbudowane na bazie Expression Tree z LINQ

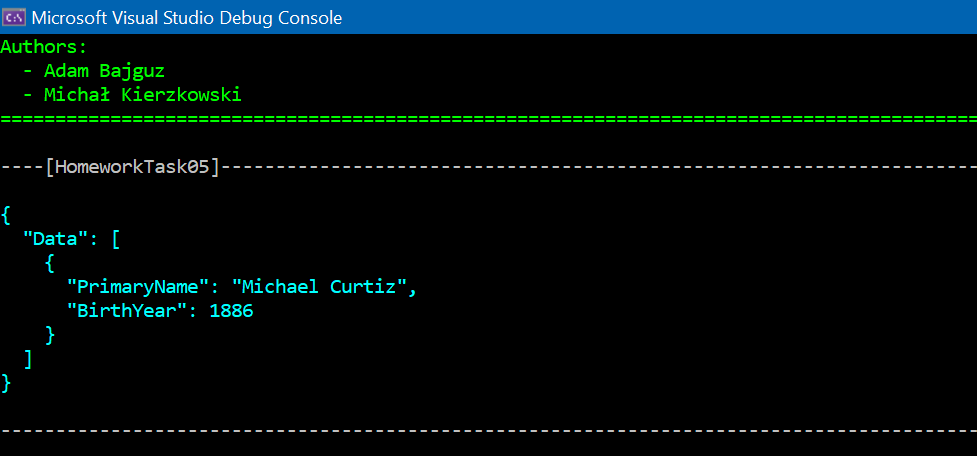
Należy zauważyć że użycie ToLower() lub ToUpper() tak naprawdę powoduje zastosowanie porównywania case insenstive, co pokazuje wpis z logów na Rys. 4.1. Parametr **i** po wyrażeniu regularnym np. /comedy/ oznacza że wielkość liter nie ma znaczenie, natomiast **s** umożliwia znakowi kropki (.) dopasowanie wszystkich znaki, w tym znaku nowej linii.

# Realizacja zadania 5

W celu realizacji zadania nr 5 utworzono kod przedstawiony na Rys. 5.1. Wynik działania tego kodu zaprezentowano na Rys. 5.2.



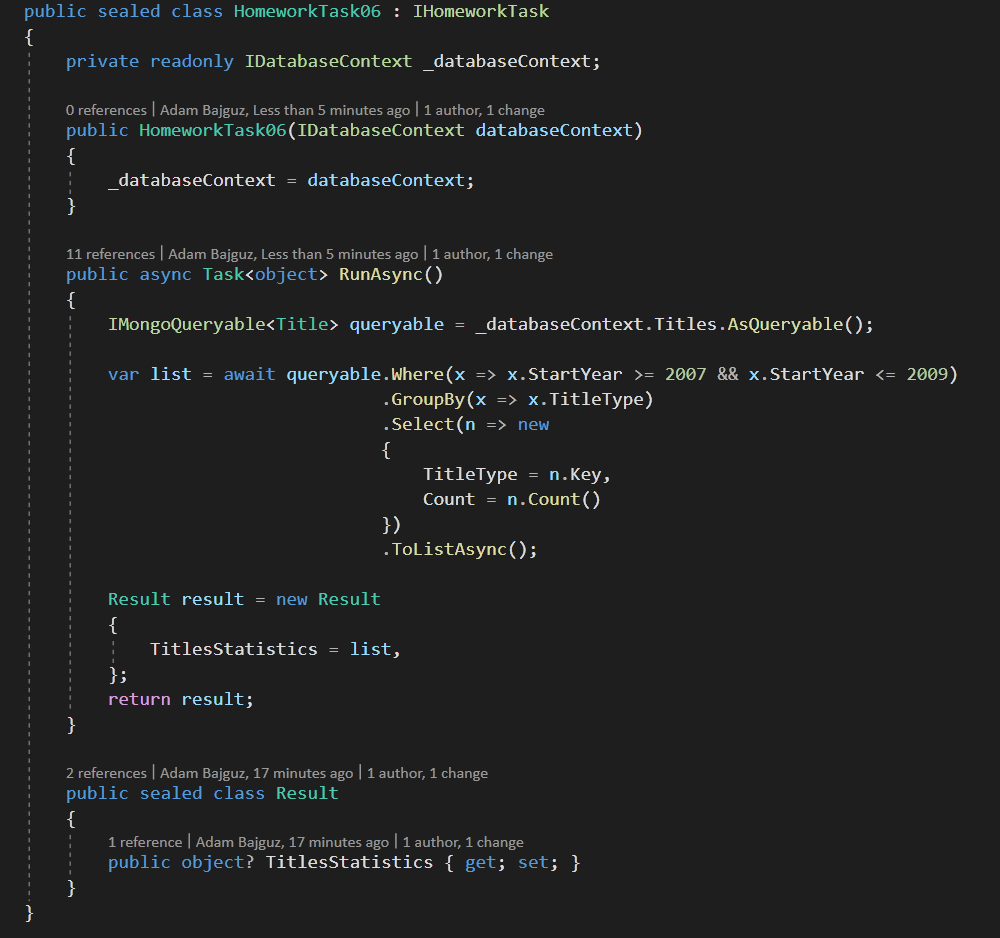
**Rys. 5.1.** Fragment najistotniejszego kodu realizującego zadanie nr 5



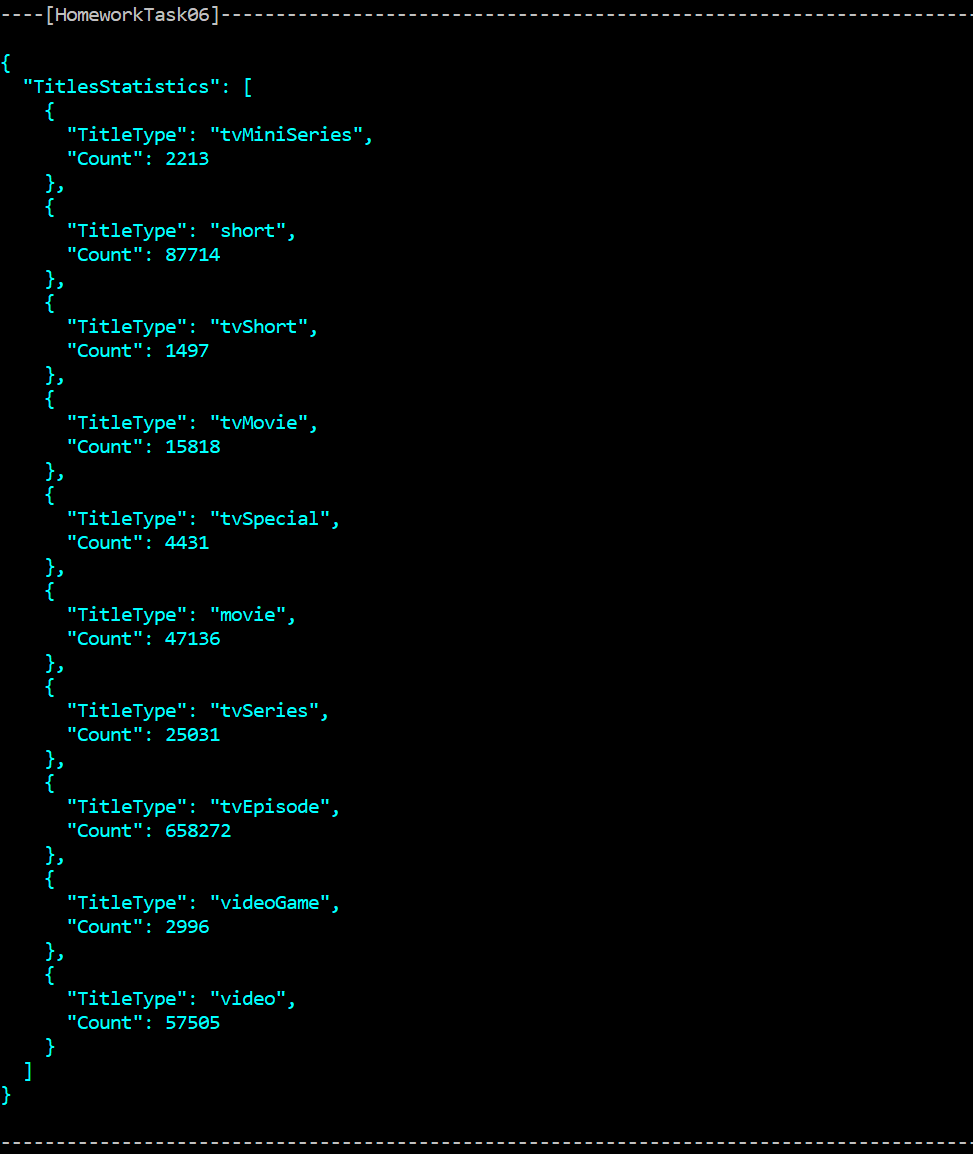
**Rys. 5.2.** Wynik działania kodu z Rys. 5.1

# Realizacja zadania 6

W celu realizacji zadania nr 6 utworzono kod przedstawiony na Rys. 6.1. Wynik działania tego kodu zaprezentowano na Rys. 6.2. Podczas realizacji zadania, ze względu na występowanie w bazie originalTitle i primaryTitle wybrano originalTile jako tytuł po którym sortowano wyniki.



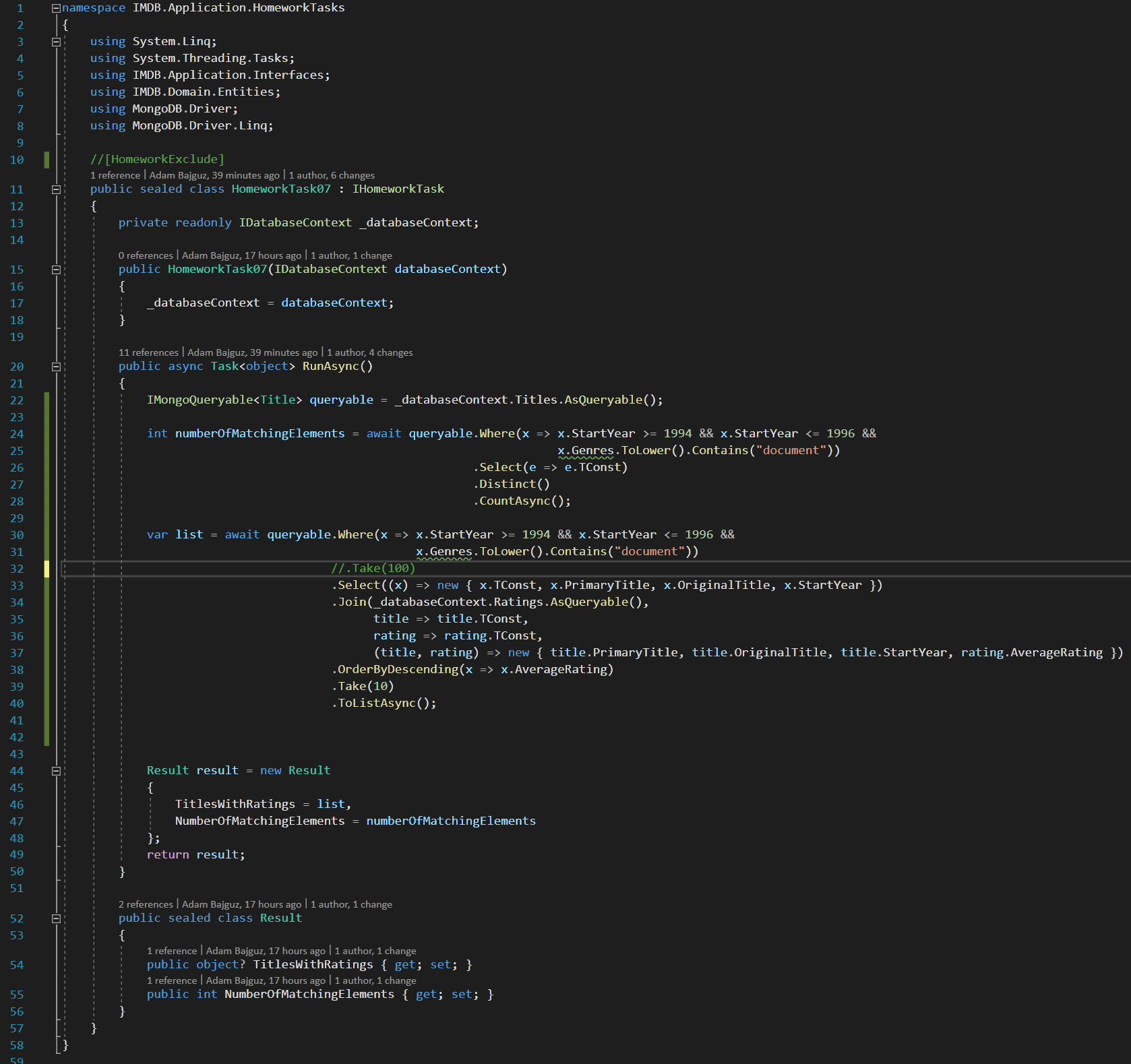
**Rys. 6.1.** Fragment najistotniejszego kodu realizującego zadanie nr 6



**Rys. 6.2.** Wynik działania kodu z Rys. 6.1

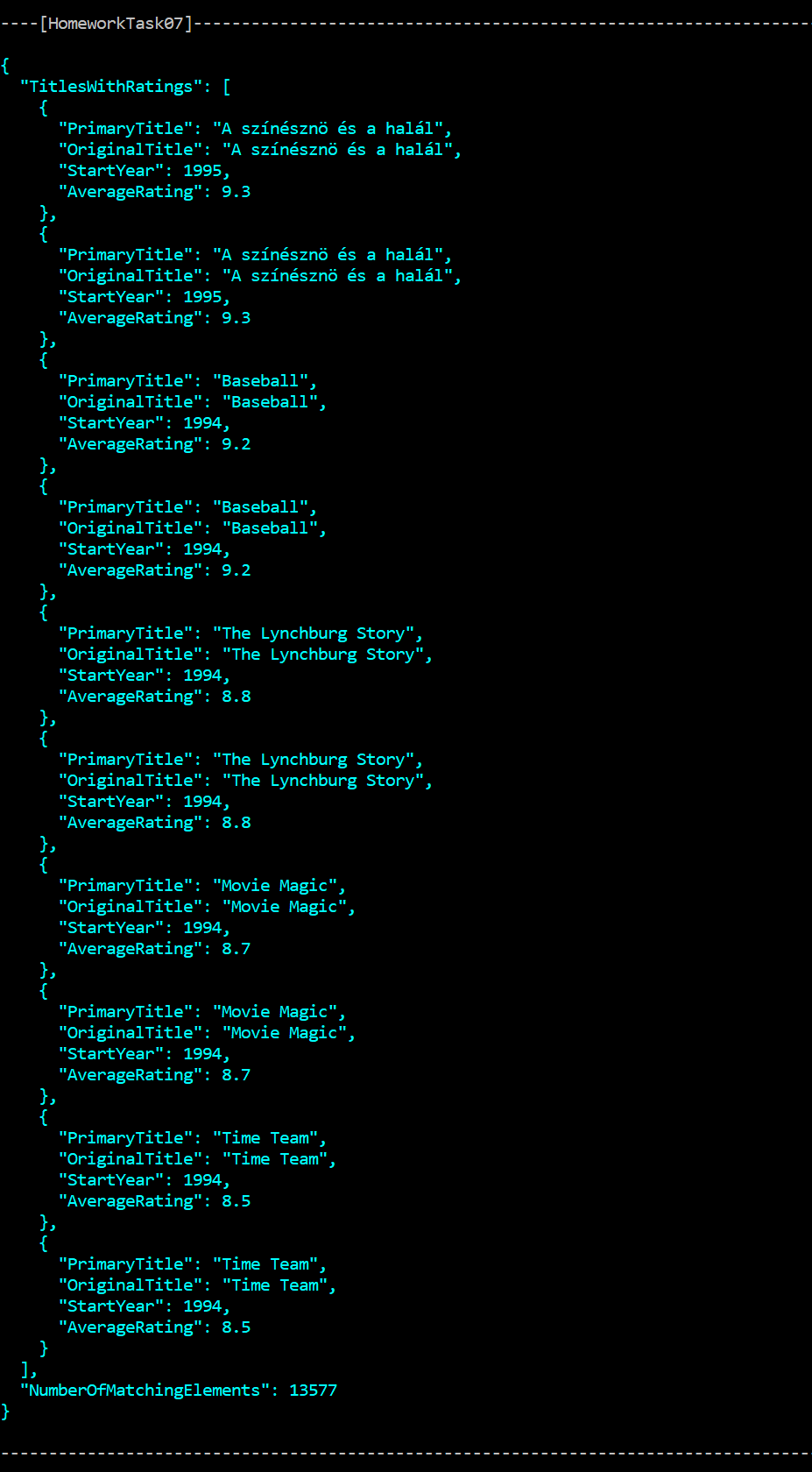
# Realizacja zadania 7

W celu realizacji zadania nr 7 utworzono kod przedstawiony na Rys. 7.1. Wynik działania tego kodu zaprezentowano na Rys. 7.2. Zapytanie zwraca 13577 dokumentów.



**Rys. 7.1.** Fragment najistotniejszego kodu realizującego zadanie nr 7

Ze względu na bardzo długi czas zajmowany prawdopodobnie przez złączenia, które w tej bazie sa oparte na wartościach tekstowych. Wynik przedstawiony na Rys. 7.2 jest oparty na 100 filmach – odkomentowania linia nr 32. Możliwym rozwiązaniem tego problemu była by konwersja pól i wartości z tekstowych na liczbowe i dodanie indeksu, jednakże polecenie nie określa czy taka operacja jest dopuszczalna. Zatem podjęto decyzję o zaprezentowaniu działania kodu na wspomnianym podzbiorze 100 filmów.



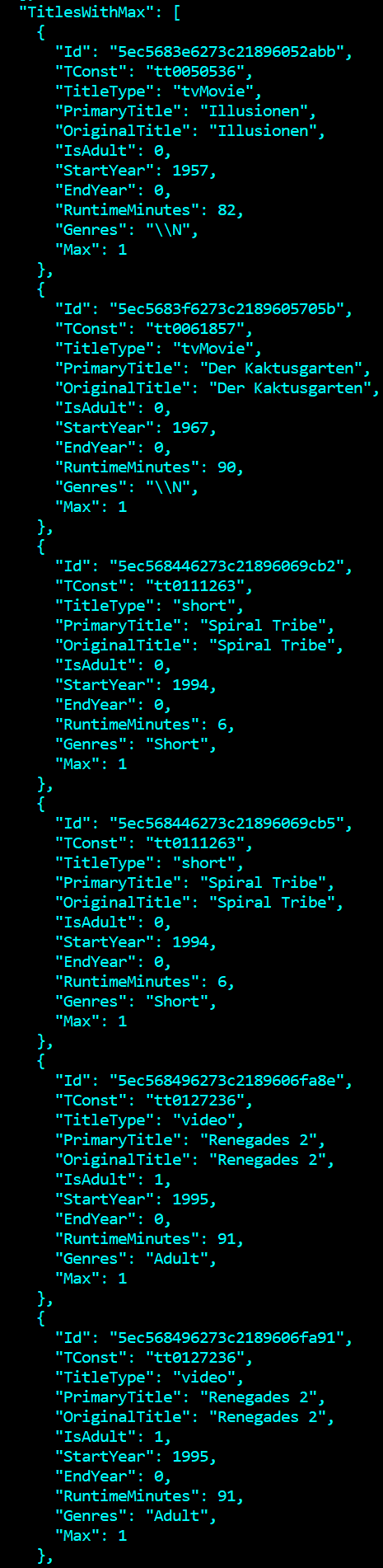
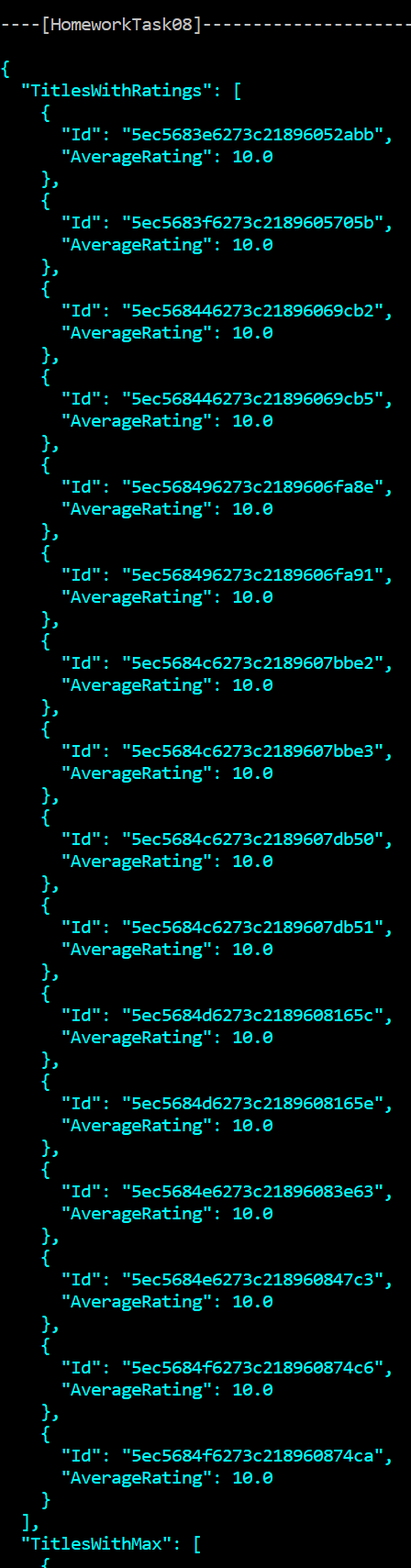
**Rys. 7.2.** Wynik działania kodu z Rys. 7.1

# Realizacja zadania 8

W celu realizacji zadania nr 8 utworzono kod przedstawiony na Rys. 8.1. Wynik działania tego kodu zaprezentowano na Rys. 8.2.



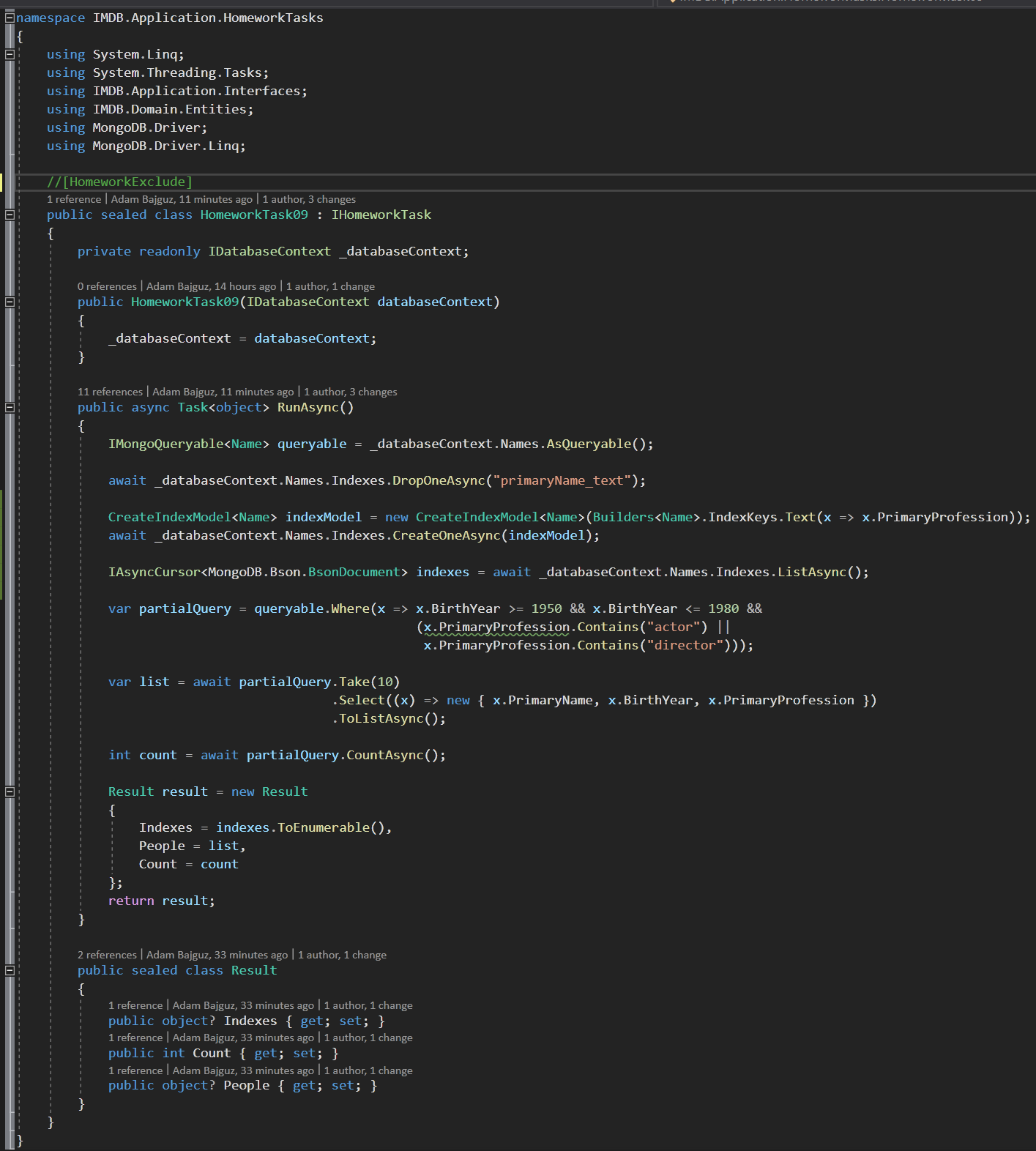
**Rys. 8.1.** Fragment najistotniejszego kodu realizującego zadanie nr 8



**Rys. 8.2.** Wynik działania kodu z Rys. 8.1 przedstawiający część danych które uległy zmianie

# Realizacja zadania 9

W celu realizacji zadania nr 9 utworzono kod przedstawiony na Rys. 9.1. Wynik działania tego kodu zaprezentowano na Rys. 9.2. Z uwagi na niejasność w polecaniu wybrano 10 dokumentów a nie 5. Zapytanie zwraca 94865 dokumentów.

****

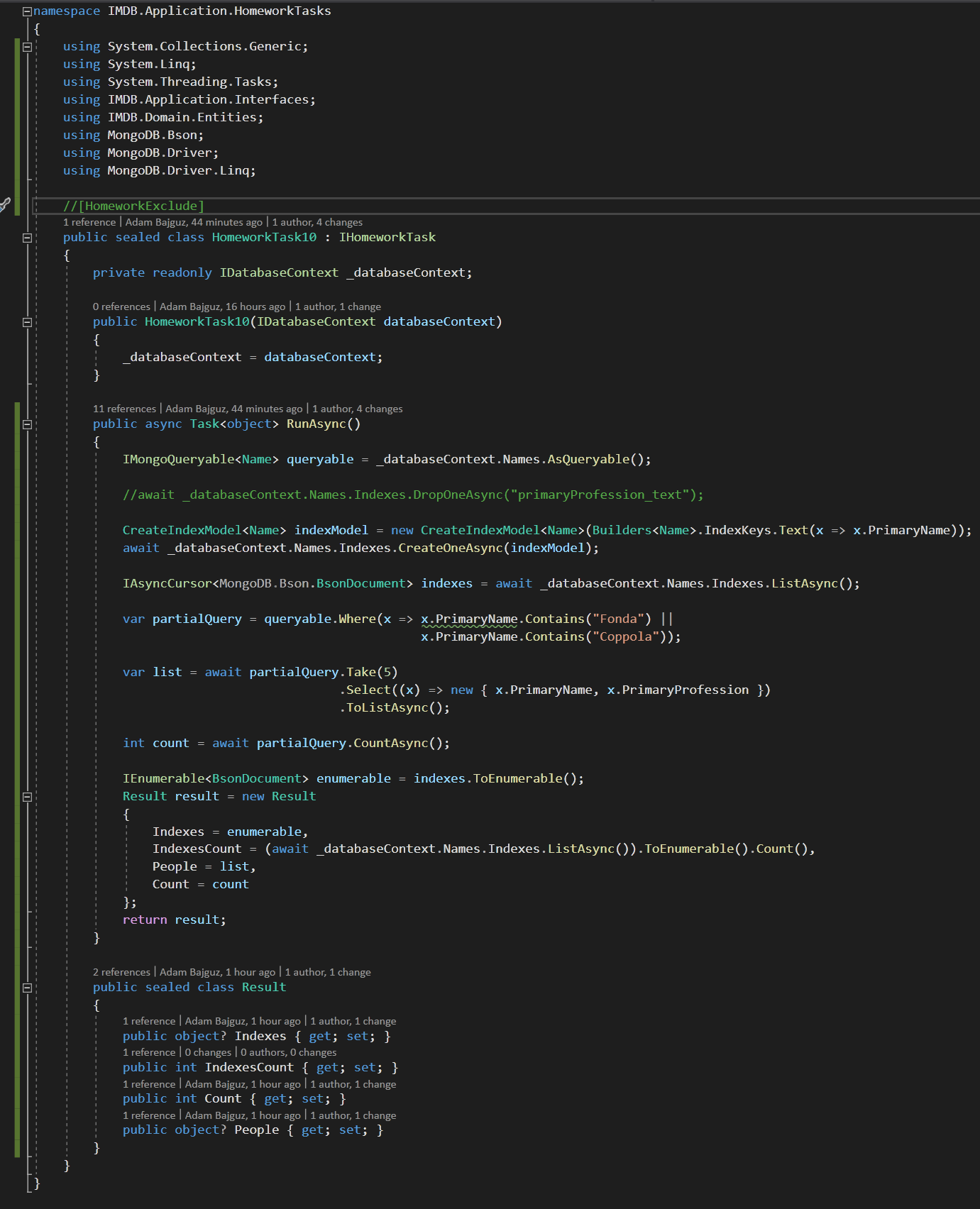
**Rys. 9.1.** Fragment najistotniejszego kodu realizującego zadanie nr 9



**Rys. 9.2.** Wynik działania kodu z Rys. 9.1

# Realizacja zadania 10

W celu realizacji zadania nr 10 utworzono kod przedstawiony na Rys. 10.1. Wynik działania tego kodu zaprezentowano na Rys. 10.2. Zapytanie zwraca 617 dokumentów, a liczba indeksów wynosi dwa.



**Rys. 10.1.** Fragment najistotniejszego kodu realizującego zadanie nr 10



**Rys. 10.2.** Wynik działania kodu z Rys. 10.1

# Wnioski

* W zwracanych danych występowały wartości „\N” w miejscu liczb całkowitych. W celu rozwiązania problemu zostały one zmapowanie na null.
* Podczas realizacji zadań zauważono że tytuły filmów są zapisane w polu primaryTitle oraz originalTitle. W zadaniu nr 3 i 7 z uwagi na niejednoznaczność treści wypisano obydwa te tytuły.
* Zastosowanie oficjalnej biblioteki do obsługi MongoDB z NuGet w konsolowej aplikacji .NET Core 3.1.4 pozwoliło na szybką realizację zadań w sposób zbliżony do tego jak taka realizacja mogła by wyglądać przy stosowaniu bazy relacyjnej i biblioteki EntityFramework lub EntityFrameworkCore.
* Zastosowanie LINQ, które oficjalna biblioteka MongoDB do .NET wspiera natywnie, pozwoliło na nałożenie składniowej abstrakcji na etapie dostępu do danych. Ponadto w przeciwieństwie do drugiej możliwej do zastosowania metody, tj. używanie metod operujących na m.in. filtrach, składnia jest prostsza i przypomina SQL, chociaż w rzeczywistości i tak zapisane wyrażenia są tłumaczone i wykonywane w języku MQL.
* Metoda Count() / CountAsync() zwracająca dokładną liczbę elementów w kolekcji wykonuje się długo, a szybkość wykonania zależy od rozmiaru kolekcji. Rozwiązaniem tego mogłoby być zastosowanie metody EstimatedCount() / EstimatedCountAsync(), lecz wówczas liczba elementów mogłaby nie odpowiadać rzeczywistej.
* Metoda EstimatedCount() ma złożoność O(1), a Count() – O(n). Jest to spowodowane metadanymi używanymi do zliczania. Zauważono że druga metoda ma znaczne zapotrzebowanie na pamięć operacyjną.
* Użycie ToLower() lub ToUpper() powoduje zastosowanie porównywania nie uwzględniającego wielkości liter. Z punktu widzenia .NET i programisty to zachowanie dziwne, gdyż zezwala na zwrócenie true przez (…).ToUpper().Contains("coppola"), chociaż Contains domyślnie w normalnym zastosowaniu wewnątrz .NET używa porównywania bajtowego z uwzględnieniem wielkości liter StringComparison.Ordinal. Oznacza to że oficjalna biblioteka MongoDB zawiera zachowania, które mogą prowadzić do błędnego wykonywania kodu, który na pozór wygląd dobrze i powinien się tłumaczyć na MQL w całkowicie inny sposób.
* Zauważono, że MongoDB może wykorzystać wszystkie rdzenie procesora na komputerze wielordzeniowym do operacji odczytu, ale dla operacji zapisu w obrębie jedno procesu MongoDB prawdopodobnie może wykorzystywać tylko jeden rdzeń.
* Zauważono, że MongoDB podczas operacji zapisuj na dużych kolekcjach ma znaczne zapotrzebowanie na RAM. Przykładowo kolekcja około 10 milionów elementów i rozmiarze około 3GB podczas wykonywania zapytania zmieniającego jedno pole powoduje użycie nawet 8GB pamięci operacyjnej. Jednakże zużycie dysku i procesora w porównaniu do pamięci operacyjnej pozostaje niskie, co może sugerować że MongoDB nie jest optymalnym systemem bazodanowym do zapisu/aktualizacji danych.
* „Złączenia” kolekcji po „kluczu” tekstowym zajmują bardzo dużo czasu i zasobów sprzętowych.