Porównanie bazy relacyjnej z bazą dokumentową MongoDB

# Wstęp

Realizowany Projekt ma na celu wskazanie różnić pomiędzy bazą relacyjną SQL, a bazą dokumentową. W tym celu została utworzona aplikacja testująca wydajność baz dla funkcji CRUD(od ang. create, read, update and delete) dla 10, 100, 1000, 10000 operacji.

|  |  |  |
| --- | --- | --- |
| **Działanie** | **Instrukcja SQL** | **MongoDB** |
| Create | INSERT | INSERT |
| Read (Retrieve) | SELECT | FIND |
| Update | UPDATE | UPDATE |
| Delete (Destroy) | DELETE | DELETE |

Aplikacja została napisana w języku C# przy wykorzystaniu frameworka ASP.NET 4.  
Do realizacji połączenia z bazą danych MongoDB wykorzystano oficjalny driver dostępny na oficjalnej stronie MongoDB. Do połączenia bazy relacyjnej z aplikacją zostało użyte wbudowane narzędzie w Visual Studio 2015

# Terminologia i pojęcia

Wiele pojęci używanych w relacyjnych bazach danych SQL ma swoje odpowiedniki w MongoDB. Poniższa tabela przedstawia kilka z nich.

|  |  |  |
| --- | --- | --- |
| **MySQL** | **Instrukcja SQL** |  |
| Table(tabela) | Collection(kolekcja) |  |
| Row(rekord) | Document(dokument) |  |
| Column(kolumna) | Field(pole) |  |
| Joins(złączenia) | Embedded documents(dokumenty osadzone), linking |  |
|  |  |  |

# Charakterystyka MongoDB

Monogo jest otwartym, nierelacyjnym systemem zarządzania bazą danych udostępnionym na otwartej licencji GNU AGPL. Charakteryzuje się dużą skalowalnością, wydajnością oraz brakiem ściśle zdefiniowanej struktury obsługiwanych baz danych. Zamiast tego dane składowane są jako dokumenty w stylu JSON, co umożliwia aplikacjom bardziej naturalne ich przetwarzanie, przy zachowaniu możliwości tworzenia hierarchii oraz indeksowania. Doskonale sprawdza się w rozwiązaniach internetowych.

Zamiast przechowywać dane w serii rekordów i tabel, jak ma to miejsce w SQL, MongoDB zawiera kolekcje i dokumenty. Kolekcję można porównać do tabeli, jednak dokument różni się od rekordu. Dokumenty ze względu na to, że nie mają ściśle określonej struktury mogę przechowywać znacznie więcej informacji.

Załóżmy, że chcemy przechowywać informację o użytkownikach i ich rolacha.

W SQL będzie to wyglądać tak.

|  |  |
| --- | --- |
| **USERS** | |
| **user\_id** | **email** |
| 51 | xxx@onet.pl |
| 52 | yyy@onet.pl |
| **ROLES** | |
| **role\_id** | **name** |
| 1 | Admin |
| 2 | Moderator |
| **USER\_ROLES** | |
| **role\_id** | **user\_id** |
| 1 | 51 |
| 2 | 51 |
| 2 | 52 |

Jak widać ustalenie ról wymaga przeprowadzenia operacji złączeń, gdy w przypadku MongoDB sprowadza się do utworzenia dokumentu o następującej postaci:

1. {
2. "id" : ObjectId("5678cdef19875672av1456avasceg"),
3. "email" : xxx@onet.pl,
4. "roles" : ["admin", "operations"]
5. }

Podejście to może rodzić pewne wątpliwości co do na przykład nadmiarowości danych, albo możliwości aktualizacji nazwy roli np. z admin na administrator. W SQL głównym celem jest przygotowanie struktury danych w taki sposób, by uniknąć redundancji danych, więc modyfikacji danych sprowadzałaby się tylko do zmiany nazwy w rekordzie tabeli **ROLES**.

Jeżeli chodzi o MongoDB głównym celem jest ułatwienie użycia danych w aplikacji. W takim przypadku zmiana nazwy roli będzie wymagała znacznie większej ilości operacji. Z założenia jednak operacja aktualizacji danych przy tym podejściu ma być rzadką operacją.

Dane w MongoDB przechowywane są w postaci BSON(Binary JSON), który nieco różni się od JSON. Jedną z różnic jest wartość **ObjectId**, która jest typem używanym w MongoDB do unikatowej identyfikacji użytkowników(odpowiednik klucza podstawowego w SQL). Wartość ObjectId nie tylko jednoznacznieidentyfikujedokument ale na jej podstawie jesteśmy wstanie określić kiedy dokument został wygenerowany.

# Zalety formatu BSON

Zalety formatu BSON w porównaniu do tabel i rekordów:

* Naturalny sposób mapowania na obiekty w językach programowania
* Osadzone dokumenty i tablice eliminują konieczność stosowania złączeń
* Brak schematu danych przekładający się na ułatwienie polimorfizmu

# MongoDB skalowanie poziome i automatyczny sharding

MongoDB obsługuję skalowanie poziome , które pozwala na dodanie kolejnych maszyn do klastra i tym samym jego skalowanie. Jest to przeciwieństwo skalowania pionowego, które sprowadza się do zwiększania mocy obliczeniowej poszczególnych komputerów.

W przypadku baz SQL są ograniczone możliwości zwiększenia wydajności bazy. W grę wchodzą: analiza i optymalizacja zapytań, zwiększenie bufora, przeniesienie danych do innych źródeł, zmniejszenie poziomu użycia serwera podczas zapytań. Wszystkie te rozwiązania wymagają dużych nakładów czasowych związanych z pracą programistów.

Jedną z możliwości zwiększenia wydajności relacyjnej bazy danych jest zastosowanie tzw. Sharding sytemu, co oznacza podział danych między wiele baz danych. W przypadku relacyjnych baz danych jest to trudne przedsięwzięcie, dodatkowo wiążę się z możliwością wykonywania zapytań dotyczących całego zbioru danych. W przypadku MongoDB problem z shardingiem nie występuję, jednak istnieją pewne ograniczenia z nim związane.

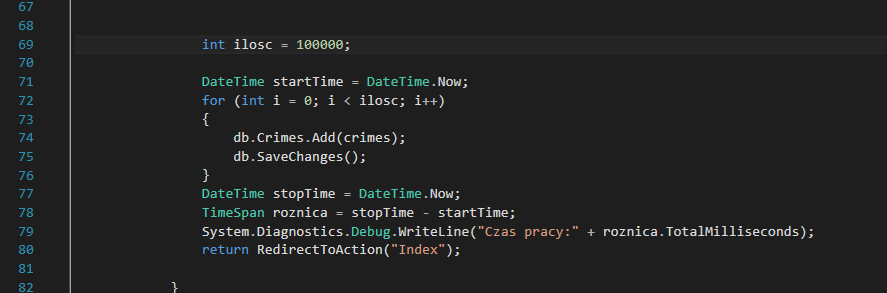
# Realizacja połączenia w aplikacji.

1. var client = **new** MongoClient(**new** MongoClientSettings {
2. Server = **new** MongoServerAddress("localhost", 3979)
3. });
4. IMongoDatabase \_database = client.GetDatabase("dvproject");

# Przykładowy test.

* 1. **Test dla MongoDB:**

1. **private** **float**[] MongoCreateTest(IMongoCollection<Crime> collection, **int**[] insertNumbers)
2. {
3. System.Diagnostics.Debug.WriteLine("\*\*\*\*\*\*\*\*\*\*CreateTest\*\*\*\*\*\*\*\*\*\*");
4. **float**[] results = **new** **float**[insertNumbers.Length];
5. **for** (**int** i = 0; i < insertNumbers.Length; i++)
6. {
7. var watch = System.Diagnostics.Stopwatch.StartNew();
8. mongoInsertOperation(collection, insertNumbers[i]);
9. watch.Stop();
10. var elapsedMs = watch.ElapsedMilliseconds;
11. results[i] = elapsedMs;
12. System.Diagnostics.Debug.WriteLine(insertNumbers[i] + "\t\t" + elapsedMs);
13. }
15. **return** results;
16. }
17. **private** **void** mongoInsertOperation(IMongoCollection<Crime> collection, **int** insertNumber)
18. {
19. **for** (var i = 0; i < insertNumber; i++)
20. {
21. collection.InsertOne(**new** Crime()
22. {
23. Id = ObjectId.GenerateNewId(),
24. DateRptd = "01/01/2016",
25. DrNo = 1,
26. DateOcc = "01/01/2016",
27. TimeOcc = 1,
28. Area = 1,
29. AreaName = "Cracov",
30. CrmCd = 1,
31. Rd = 1,
32. CrmCdDesc = "1",
33. Status = "fail",
34. StatusDesc = "fail",
35. Location = "Cracov",
36. CrossStreeet = "Blabla",
37. Location1 = "Cracov1"
38. });
39. }
40. }
    1. **Test dla bazy relacyjnej:**

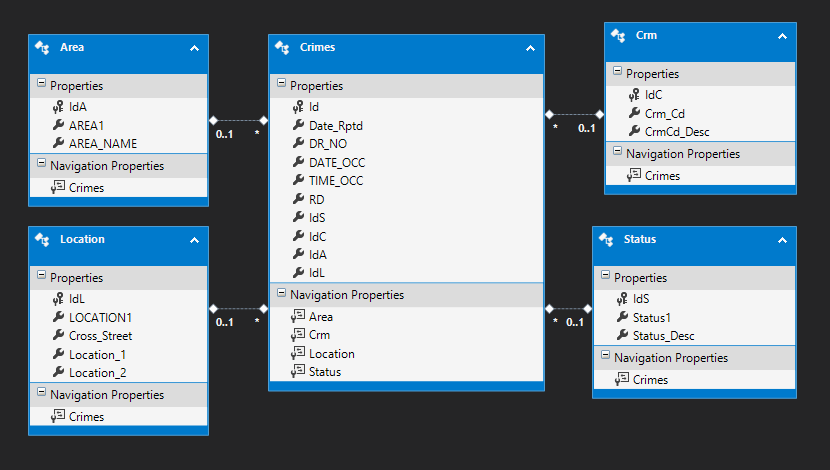


# 8. Kod JavaSript odpowiedzialny za rysowanie wykresów.

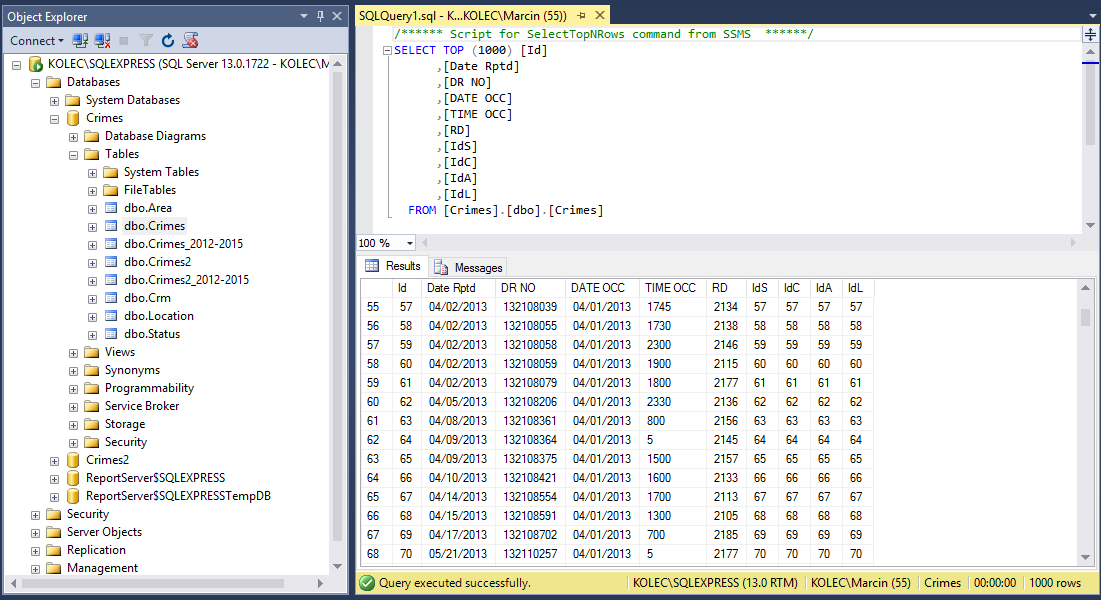
1. google.charts.load('current', { packages: ['corechart', 'bar'] });
2. **var** results = **null**;
3. $("#runTest").click(**function** () {
4. $.ajax({
5. url: "../../CRUD/RunCRUDTest",
6. type: "GET",
7. data: **null**,
8. success: **function** (response) {
9. results = response;
10. google.charts.setOnLoadCallback(drawCharts);
11. alert('OK');
12. },
13. error: **function** (response) {
14. alert('BAD');
15. }
16. });
17. });
19. $("#checkItOut").click(**function** () {
20. $.ajax({
21. url: "../../CRUD/Count?value=" + $("#arename").val(),
22. type: "GET",
23. data: **null**,
24. success: **function** (response) {
25. results = response;
26. $("#crimes\_number").text(results);
27. alert('OK');
28. },
29. error: **function** (response) {
30. alert('BAD');
31. }
32. });
33. });

36. **function** drawCharts() {
37. //Create Test
38. drawOneChart('Create', 'create\_div', results.operationsNumbers, results.mongoCreate, results.mySQLCreate);
39. //Read Test
40. drawOneChart('Read', 'read\_div', results.operationsNumbers, results.mongoRead, results.mySQLRead);
41. //Update Test
42. drawOneChart('Update', 'update\_div', results.operationsNumbers, results.mongoUpdate, results.mySQLCreate);
43. //Delete Test
44. drawOneChart('Delete', 'delete\_div', results.operationsNumbers, results.mongoDelete, results.mySQLCreate);
45. }
47. **function** drawOneChart(testName, divId, opeartionNumber, mongoResults, mySQLResults) {
48. **var** data = **new** google.visualization.DataTable();
49. data.addColumn('string', 'Operation Number');
50. data.addColumn('number', 'Mongo');
51. data.addColumn('number', 'MySQL');
53. data.addRows([
54. [{ v: opeartionNumber[0].toString(), f: 'Time' }, mongoResults[0], mySQLResults[0]],
55. [{ v: opeartionNumber[1].toString(), f: 'Time' }, mongoResults[1], mySQLResults[1]],
56. [{ v: opeartionNumber[2].toString(), f: 'Time' }, mongoResults[2], mySQLResults[2]],
57. [{ v: opeartionNumber[3].toString(), f: 'Time' }, mongoResults[3], mySQLResults[3]],
58. [{ v: opeartionNumber[4].toString(), f: 'Time' }, mongoResults[4], mySQLResults[4]]
59. //[{v: "4", f: '11 am'}, results.mongoCreate[3], 2.25]
60. ]);
62. **var** optionsCreate = {
63. chart: {
64. title: 'Speed test '+ testName + ' operation',
65. subtitle: ''
66. },
67. hAxis: {
68. title: 'Number of operations',
69. format: 'h:mm a',
70. viewWindow: {
71. min: [7, 30, 0],
72. max: [17, 30, 0]
73. }
74. },
75. vAxis: {
76. title: 'Rating (scale of 1-10)'
77. }
78. };
80. **var** material = **new** google.charts.Bar(document.getElementById(divId));
81. material.draw(data, optionsCreate);
82. }

**9. Relacyjna baza danych**



**Podgląd wyników w programie Microsoft SQL Server 2016**



**Encje:**

**ENC/1 Crimes**

Sematyka: Przechowuje dane na temat Przestępstw w Ameryce w latach 2012-2015

|  |  |  |  |
| --- | --- | --- | --- |
| **Nazwa atrybutu** | **Opis atrybutu** | **Typ** | **OBL(+) OPC(-)** |
| Id | Identyfikator | Int | + |
| Date\_Rpd | Data zaraportowania | Date | + |
| DR\_NO | Numer zdarzenia | Int | + |
| DATE\_OCC | Data zdarzenia | Date | + |
| TIME\_OCC | Godzina zdarzenia | Date | + |
| RD | Data produkcji danej serii | Data | + |
| IdS | Klucz obcy Statusu | Int | - |
| IdC | Klucz obcy Crm | Int | - |
| IdA | Klucz obcy Area | Int | - |
| IdL | Klucz obcy Location | Int | - |

*Klucz główny*: Id

**ENC/2 Crm**

Sematyka: Przechowuje dane na temat Przestępstw wraz z ich opisem

|  |  |  |  |
| --- | --- | --- | --- |
| **Nazwa atrybutu** | **Opis atrybutu** | **Typ** | **OBL(+) OPC(-)** |
| IdC | Identyfikator Crm | Int | + |
| Crm\_Cd | Numer przestępstwa | Varchar(max) | + |
| Crm\_Cd\_Desc | Opis przestępstwa | Varchar(max) | + |

*Klucz główny: IdC*

**ENC/3 Status**

Sematyka: Przechowuje dane na temat Statusu Przestępstw

|  |  |  |  |
| --- | --- | --- | --- |
| **Nazwa atrybutu** | **Opis atrybutu** | **Typ** | **OBL(+) OPC(-)** |
| IdS | Identyfikator Status | Int | + |
| Status | Status | Varchar(50) | + |
| Status\_Desc | Opis Statusu | Varchar(max) | + |

*Klucz główny: IdS*

**ENC/4 Location**

Sematyka: Przechowuje dane na temat Lokalizacji Przestępstw

|  |  |  |  |
| --- | --- | --- | --- |
| **Nazwa atrybutu** | **Opis atrybutu** | **Typ** | **OBL(+) OPC(-)** |
| IdL | Identyfikator Location | Int | + |
| LOCATION | Lokacja | Varchar(max) | + |
| Cross\_Street | Adres | Varchar(max) | + |
| Location\_1 | Współrzędna 1 | Double | + |
| Location\_2 | Współrzędna 2 | Double | + |

*Klucz główny: IdL*

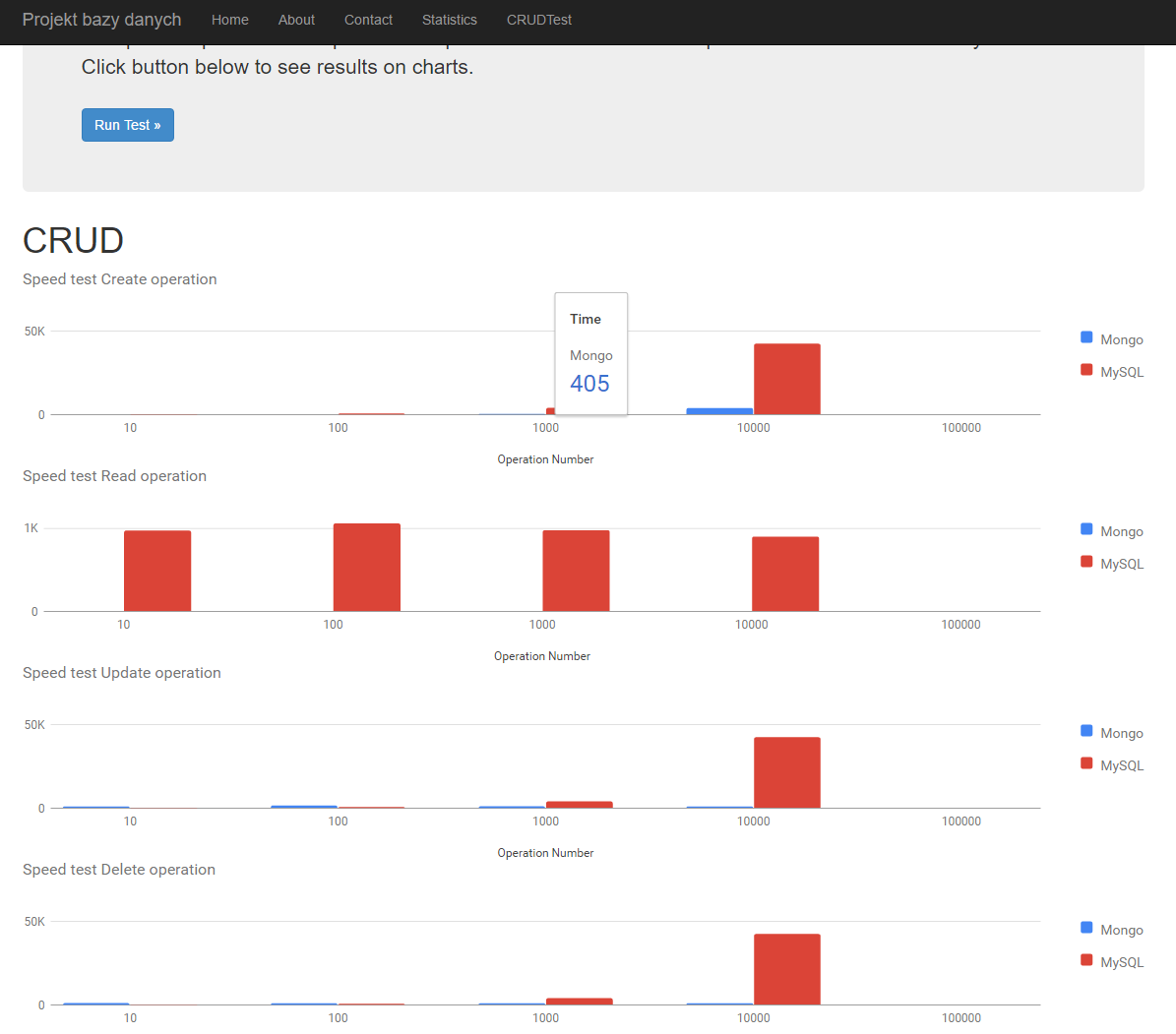
**ENC/5 Area**

Sematyka: Przechowuje dane na temat Stanów w jakich dochodziło do przestępstw

|  |  |  |  |
| --- | --- | --- | --- |
| **Nazwa atrybutu** | **Opis atrybutu** | **Typ** | **OBL(+) OPC(-)** |
| IdA | Identyfikator Area | Int | + |
| AREA | Numer Area | Varchar(max) | + |
| AREA\_NAME | Nazwa Area | Varchar(max) | + |

*Klucz główny: IdA*

# Wyniki wygenerowane przez program



# Wnioski