POLSKIE TOWARZYSTWO INFORMACJI PRZESTRZENNEJ
ROCZNIKI GEOMATYKI 2018 ❍ Tom XVI ❍ Zeszyt 1(80): 55–69

Przegląd możliwości zastosowania wybranych baz danych NoSQL do zarządzania danymi przestrzennymi

An overview of possibilities of using selected NoSQL databases to manage spatial data

Michał Wyszomirski

Politechnika Warszawska, Wydział Geodezji i Kartografii, Zakład Kartografii

Słowa kluczowe: bazy danych, dane przestrzenne, NoSQL
Keywords: databases, spatial data, NoSQL

Geneza powstania baz NoSQL

Trudno jednoznacznie stwierdzić, od czego rozpoczęła się historia baz określanych nazwą NoSQL. Niektóre źródła (Sadalage, Fowler, 2012) podają, iż pierwszą bazą NoSQL była baza danych operująca na plikach tekstowych. Opracował ją Carlo Strozzi w 1998 roku. Dane zapisywane były w wierszach odpowiadających rekordom w relacyjnych bazach danych, a poszczególne wartości oddzielane były tabulatorem. Mechanizm dostępu do danych realizowany był przez powłokę[1] systemu UNIX (Sadalage, Fowler, 2012). Aby zrozumieć ideę i potrzebę wykorzystania baz NoSQL, trzeba najpierw określić przyczyny poszukiwania innych rozwiązań niż powszechnie znane i sprawdzone relacyjne bazy danych bądź rozwijające się bazy danych obiektowych.

Twórca teorii relacyjnych baz danych Edgar Frank Codd opublikował swoje opracowanie *A Relational Model of Data for Large Shared Data Banks* w 1970 roku. Zaproponował w nim model relacyjny organizacji danych bazujący na matematycznej teorii mnogości, w szczególności na pojęciu relacji. W modelu relacyjnym dane grupowane miały być w relacje reprezentowane przez tablice. Relacje są zbiorem rekordów o identycznej strukturze wewnętrznie powiązanych za pomocą związków zachodzących pomiędzy danymi. W czasie powstawania propozycji Codda rozwijały się już dwa inne modele danych: model sieciowy oraz model hierarchiczny. W 1973 roku firma IBM przygotowała System R będący pierwszą implementacją modelu relacyjnego i języka SEQUEL, znanego później jako SQL. W roku 1979 firma Relational Software wypuściła na rynek pierwszy komercyjny relacyjny

[1] Powłoka systemowa – program pełniący rolę pośrednika pomiędzy systemem operacyjnym lub aplikacją a użytkownikiem, pozwalający wprowadzać polecenia i zwracający wyniki działania programów.

system zarządzania bazą danych Oracle v2. W tym czasie wzorce użytkowania aplikacji były doskonale znane. Obciążenie serwerów wymagane przez relacyjną bazę danych można było określić i oszacować. Model danych był znormalizowany, jak zaleca teoria relacyjna. Relacyjne bazy danych zapewniały wiele korzyści, takich jak: wsparcie transakcji, spójność danych i izolacja. Relacyjne bazy danych idealnie nadają się do tych celów. Dlatego też nie jest trudno zrozumieć, dlaczego relacyjna baza danych była tak popularna i dlaczego jest *de facto* standardem dla trwałych magazynów danych w rozwoju aplikacji (Kan, 2014).

Mimo wielkiej popularności, jaką zdobyły relacyjne bazy danych, użytkownicy dostrzegali ich ograniczenia, w szczególności zastosowanie prostych typów danych, co w znacznym stopniu utrudniało obsługę danych zmieniających się w czasie (ang. *temporal data*), danych przestrzennych (ang. *spatial data*), danych multimedialnych (ang. *multimedia data*) i danych o nieuporządkowanej strukturze (ang. *unstructured data*). Ponadto zastosowanie baz relacyjnych w nowoczesnych rozwiązaniach aplikacyjnych ujawniło dodatkowe, niżej omówione problemy (Strauch, n.d.).

Problem współbieżności występuje, gdy wielu użytkowników pracuje na tych samych danych oglądając je i modyfikując. Transakcje pozwalają nadzorować spójność danych w bazie danych, lecz po stronie aplikacji również trzeba kontrolować i utrzymywać aktualność i spójność danych, na przykład problem rezerwacji jednego pokoju przez dwóch klientów w aplikacji internetowej (Sadalage, Fowler, 2012).

Problem integracji występuje, gdy baza danych jest bazą integrującą, to znaczy wiele aplikacji używa tych samych danych. W celu zapewnienia dostępu do danych w sposób wygodny dla każdej z aplikacji z niej korzystającej, struktura bazy danych staje się skomplikowana. Jeżeli jedna aplikacja wymaga dokonania zmian w sposobie przechowywania danych, to musi to zostać uzgodnione z twórcami pozostałych aplikacji korzystających z tych samych danych. Rozwiązaniem problemu jest prowadzenie niezależnych struktur dla każdej z aplikacji, ale to z kolei może prowadzić do redundancji danych (Sadalage, Fowler, 2012).

Problem impedancji polega na niezgodności modelu danych w bazie z modelem danych w aplikacji. Dane zapisane w strukturach danych aplikacji muszą być konwertowane do postaci umożliwiającej ich zapis w bazie, między innymi konieczność normalizacji danych. Taka operacja konsumuje zasoby serwera, wydłużając czas zapisu i odczytu danych (Sadalage, Fowler, 2012).

Problem skalowania – bardzo duże zasoby danych wymagają zastosowania klastrów do przechowywania danych na wielu współpracujących ze sobą serwerach. Taka architektura wymaga zastosowania skalowania „w bok" (ang. *scale out*, *horizontal scaling*). Bazy relacyjne umożliwiają jedynie skalowanie „w górę" (ang. *scale up*, *vertical scaling*) przez dodanie zasobów do serwera (kolejnych rdzeni procesorów i pamięci operacyjnej). Wyjątkami są tu MySQL posiadający cechy pozwalające na klastrowanie serwerów oraz Oracle RAC, pozwalający na pracę wielu serwerów korzystających z tych samych plików z danymi. To rozwiązanie opiera się jednak na pojedynczym elemencie podatności na awarię (ang. *single point of failure*), ponieważ wszystkie serwery używają tych samych plików danych (Yarabarla, 2017).

Powyższe ograniczenia były między innymi przyczyną poszukiwania rozwiązań alternatywnych do relacyjnego modelu danych. Rozpowszechnienie Internetu i liczne aplikacje działające w sieci spowodowały, że coraz trudniej jest przewidzieć liczbę użytkowników takiej aplikacji, sposobów jej wykorzystania, generowanego przez nią obciążenia systemu oraz niezbędnego poziomu elastyczności modelu danych. Typowe przykłady tego typu aplikacji internetowych to globalne witryny e-commerce, serwisy społecznościowe lub serwisy wideo, które w bardzo krótkim czasie generowały ogromną ilość danych. Należy również zauważyć, że dane generowane przez te aplikacje są często tylko częściowo ustrukturalizowane, a często całkowicie pozbawione struktury. Ponieważ relacyjne bazy danych są *de facto* standardem, programiści i architekci systemów informatycznych muszą je modyfikować, aby mogły obsługiwać wymienione aplikacje internetowe. Stało się oczywiste, że należy rozważyć inny rodzaj technologii umożliwiający pokonanie nowych wyzwań (Kan, 2014). Odpowiedzią na te potrzeby było na przykład opracowanie przez firmę Google rozwiązania o nazwie BigTable. Podobnie firma Amazon przygotowała do obsługi swojego sklepu internetowego własną bazę o nazwie Dynamo. Oba rozwiązania nie używają modelu relacyjnego. Uznaje się, że konferencja zorganizowana przez Johna Oskarsona 11 czerwca 2009 roku w San Francisco zapoczątkowała popularność systemów NoSQL. Celem konferencji było zaprezentowanie rozwiązań inspirowanych przez BigTable i Dynamo. Tytuł konferencji „NoSQL Meetup" nawiązywał do otwartych, rozproszonych, nierelacyjnych baz danych. Skrót „NoSQL" w tytule konferencji pierwotnie oznaczał „Not SQL" (nie SQL), później rozwinięcie skrótu zmieniono na „Not only SQL" (nie tylko SQL). Przy czym skrót SQL odnosił się nie do strukturalnego języka zapytań SQL, lecz raczej był synonimem relacyjnych baz danych (Tiwari, 2011). Od tego czasu termin NoSQL obejmuje otwarte, rozproszone, nierelacyjne bazy danych.

Cechy baz NoSQL

Bazy NoSQL nie spełniają podstawowej cechy relacyjnych baz danych związanej z transakcjami (transakcja to zbiór operacji na bazie danych, które stanowią logiczną całość i jako takie powinny być wszystkie wykonane lub wszystkie odrzucone), to jest aksjomatu ACID (ang. *Atomicity*, *Consistency*, *Isolation*, *Durability* – niepodzielność, spójność, izolacja, trwałość), stanowiącego zbiór właściwości gwarantujących poprawne przetwarzanie transakcji w bazach danych (Celko, 2014). Wymienione właściwości oznaczają:

- niepodzielność – każda transakcja albo zostanie wykonana w całości, albo w ogóle,
- spójność – po wykonaniu transakcji system będzie spójny, czyli nie zostaną naruszone zasady integralności,
- izolację – jeśli dwie transakcje wykonują się współbieżnie, to nie widzą wprowadzanych przez siebie zmian,
- trwałość – po wykonaniu transakcji pozostanie utrwalona nawet w przypadku utraty zasilania, awarii lub błędów.

Cechy baz danych typu NoSQL opisuje aksjomat CAP (*Consistency*, *Availability*, *Partition tolerance* – spójność, dostępność, tolerancja na partycjonowanie) stanowiący zbiór podstawowych wymagań opisujących dowolny system rozproszony (McCreary, Kelly, 2014):

- spójność – wszystkie serwery w systemie będą miały takie same dane, dzięki czemu użytkownicy otrzymają tę samą kopię niezależnie od tego, który serwer odpowiada na ich żądanie,

- dostępność – system zawsze reaguje na żądanie,
- tolerancja na podział na partycje – system działa w całości, nawet jeśli poszczególne serwery nie działają lub nie mogą być odpytane.

Aksjomat CAP zakłada, że możliwe jest spełnienie przez bazę danych tylko dowolnych dwóch z trzech wyżej wymienionych cech (Vaish, 2013). W rozproszonych bazach danych możliwe są tylko dwie opcje: AP (ciągła dostępność i tolerancja na podział, bez zapewnienia spójności danych) lub CP (spójność i tolerancja na podział, bez zapewnienia ciągłej dostępności danych), ponieważ w przypadku, gdy nie ma tolerancji na partycjonowanie nie jest to niezawodna rozproszona baza danych. Architekci systemów opartych na bazach danych typu NoSQL muszą, dla przypadku awarii sieci, wybrać jedno z dwóch rozwiązań:

- baza danych powinna odpowiadać na zapytania, ale z udostępnieniem prawdopodobnie starszych lub złych danych – w tym przypadku odpowiednim rozwiązaniem jest zastosowanie modelu AP,
- baza danych, gdy nie można uzyskać najnowszej wersji danych, powinna po prostu przestać odpowiadać – w tym przypadku odpowiednim rozwiązaniem jest model CP.

Analizując relacyjne bazy danych pod względem zgodności z aksjomatem CAP można powiedzieć, że pracują one w modelu AC (dostępność i spójność bez tolerancji na partycjonowanie).

Typy baz NoSQL

Istotną cechą charakterystyczną rozróżniającą dostępne obecnie rozwiązania typu NoSQL jest zastosowany model danych. Biorąc pod uwagę ten aspekt można wyróżnić bazy zorientowane na agregacje (model klucz-wartość, model dokumentowy i model rodziny kolumn), bazy zorientowane na relacje (model grafowy) oraz bazy bezmodelowe. Agregacja w przypadku baz NoSQL rozumiana jest jako kolekcja obiektów traktowanych jako całość – jednostka przeznaczona do manipulacji i pozwalająca zachować spójność danych. Operowanie na agregacjach ułatwia pracę w klastrach, ponieważ agregacja jest podstawową jednostką przetwarzaną w procesie replikacji i współdzielenia danych.

Model klucz-wartość. Pierwszym i najprostszym rodzajem baz zorientowanych na agregacje jest baza wykorzystująca model klucz-wartość (*key-value*). Baza przechowuje dane w postaci klucza i przypisanej do niego wartości (Das, 2015). Baza obsługuje operacje zapisu danych przez podanie klucza i wartości oraz odczytu wartości poprzez podanie jej klucza. Zapis pary klucz-wartość, gdzie kluczem jest numer działki ewidencyjnej (121701_1.0172.11229), a wartością jej powierzchnia (198 821 m^2), w takim przypadku wygląda w sposób następujący:

```
SET 121701_1.0172.11229 198821
```

Model dokumentowy. Kolejnym modelem danych zastosowanym w bazach zorientowanych na agregacje jest model dokumentowy. Model ten jest bardzo podobny do modelu klucz-wartość. Kluczem może być dowolny ciąg znaków, wartością jest dokument w formacie XML lub JSON. Dokumenty to samoopisujące się hierarchiczne struktury drzewiaste, które mogą się składać z wartości złożonych (tablic, zbiorów i kolekcji) lub wartości skalarnych. Ponieważ w bazach dokumentowych istnieje możliwość wyszukania dokumentu używa-

jąc jednej z wartości w nim zapisanych, zamiast podawania klucza, zazwyczaj wykorzystuje się klucze automatycznie generowane przez system. Zapis rekordu w takim przypadku wygląda w sposób następujący (przykład zapisania danych dotyczących działki ewidencyjnej):

```
db.collection.insertOne(
  {Numer: „121701_1.0172.11229”,
   Powierzchnia: 198821,
   Województwo: „małopolskie”,
   Powiat: „tatrzański”,
   Gmina: „Zakopane”,
   Obręb: „172”}
)
```

Model grafowy. Wśród baz zorientowanych na relacje najpopularniejszy jest model grafowy. W tym modelu baza składa się z węzłów, odpowiadających obiektom oraz z krawędzi łączących węzły. Zapis danych w bazie grafowej wygląda w sposób następujący (przykład zapisania działki i obrębu oraz utworzenia krawędzi łączącej oba obiekty):

```
// utworzenie działki
CREATE(d:Dzialka {Numer: „121701_1.0172.11229”, Powierzchnia:
198821});
// utworzenie obrębu
CREATE(o:Obreb {Numer: „172”, Gmina: „Zakopane”});
// wyszukanie działki i obrębu według ich numerów
MATCH(d:Dzialka {Numer: „121701_1.0172.11229”}),(o:Obreb {Numer:
„172”});
// utworzenie krawędzi typu ZAWIERA
CREATE (o)-[:ZAWIERA]->(d);
```

Dane przestrzenne w bazach NoSQL

Systemy zarządzania relacyjnymi bazami danych są od dawna skutecznie wykorzystywane w systemach GIS. Zwykle dane geoprzestrzenne mają stały schemat bazy danych, jednak coraz częściej napotykamy na dane zapisane w inny sposób. Przykładem mogą być obserwacje i pomiary z różnych rodzajów sensorów, dane przestrzenne gromadzone przez usługi LBS (ang. *Location Based Services*), dane o lokalizacji pochodzące z sieci społecznościowych, dane ze skaningu laserowego lub dane teledetekcyjne. W tym przypadku mamy do czynienia zarówno z danymi o częściowo ustalonej strukturze, jak i danym bez zdefiniowanej ściśle struktury. Poza tym mamy w tym przypadku do czynienia z ogromnymi zbiorami danych (ang. *big data*). W niektórych sytuacjach, gdy wymagana jest wysoka dostępność i skalowalność zastosowanie systemów zarządzania relacyjnymi bazami danych jest bardzo kosztowne, nawet jeśli te dane mają tradycyjny stały schemat zdefiniowany *a priori.* W tych sytuacjach bazy danych NoSQL mogą zapewnić bardzo łatwo dostępny i skalowalny sposób skutecznego zarządzania częściowo strukturyzowanymi lub niestrukturalnymi danymi geoprzestrzennymi (Amirian, Winstanley, Basiri, 2013). Bazy NoSQL początkowo skupiały się oczywiście na innych zastosowaniach niż przechowywanie danych geoprzestrzennych. Jednak z czasem zaczęto wprowadzać do nich możliwości operacji na tego typu danych.

Bazy NoSQL nie powinny być postrzegane jako konkurencja dla baz relacyjnych. Cechy baz NoSQL, takie jak: skalowanie horyzontalne, eliminacja problemu impedancji poprzez zastosowanie złożonych typów danych, bądź eliminacja problemu współbieżności przez zastosowanie architektury rozproszonej, predestynują je do przechowywania bardzo dużych zbiorów danych o skomplikowanej i często nieuporządkowanej strukturze, takich jak: dane BIM (*Building Information Model*) czy IoT (*Internet of Things*), przy jednoczesnym założeniu znacznej liczby użytkowników jednocześnie korzystających z bazy danych. Celem artykułu jest zasygnalizowanie możliwości zastosowania baz NoSQL jako mechanizmu wspomagającego przechowywanie i przetwarzanie danych przestrzennych pochodzących na przykład z systemów BIM i IoT w sytuacji, gdy ich ilość przekracza możliwości obsługi w bazach relacyjnych.

W dalszej części artykułu omówione zostaną metody przechowywania i analizy danych przestrzennych dostępne w kilku bazach danych zaliczanych do NoSQL. Do analizy zostały wybrane najpopularniejsze obecnie produkty. Wybór oparto o dane z portalu db-engines.com, który prowadzi comiesięczne badania dotyczące popularności poszczególnych systemów zarządzania bazami danych. Ranking obejmuje zarówno systemy zarządzania relacyjnymi bazami danych, jak i bazy NoSQL. Metoda pomiaru obejmuje analizę liczby stron internetowych dotyczących poszczególnych systemów baz danych znajdowanych przez wyszukiwarki Google, Bing i Yandex, częstotliwość wyszukiwania nazw poszczególnych systemów podawaną przez Google Trends, liczbę dyskusji technicznych na portalach Stack Overflow i DBA Stack Exchange, liczbę ofert pracy dotyczących poszczególnych systemów na portalach Indeed i Simply Hired, liczbę profili na portalach LinkedIn i Upwork, w których są odwołania do poszczególnych systemów zarządzania bazami danych oraz liczbę wiadomości

Tabela. Zestawienie popularności systemów zarządzania bazami danych we wrześniu 2017 roku według portalu db-engines.com

Pozycja w rankingu	Baza	Typ	Indeks
1	Oracle	Baza relacyjna	1359.09
2	MySQL	Baza relacyjna	1312.61
3	Microsoft SQL Server	Baza relacyjna	1212.54
4	PostgreSQL	Baza relacyjna	372.36
5	MongoDB	Baza dokumentowa (NoSQL)	332.73
6	DB2	Baza relacyjna	198.34
7	Microsoft Access	Baza relacyjna	128.81
8	Cassandra	Baza rodziny kolumn (NoSQL)	126.20
9	Redis	Baza klucz-wartość (NoSQL)	120.41
…	…	…	…
21	Neo4j	Baza grafowa (NoSQL)	38.42
...	...	...	...

na portalu Twitter odnoszących się do poszczególnych systemów. We wrześniu 2017 roku pierwsze trzy miejsca zajmowały odpowiednio Oracle, MySQL i SQL Server. Najwyżej notowaną bazą NoSQL była MongoDB – baza oparta o model dokumentowy (piąte miejsce w rankingu). Na miejscu ósmym znajdowała się baza Cassandra, oparta o model „rodziny kolumn" a na dziewiątym baza Redis oparta o model typu klucz-wartość. Na miejscu dwudziestym pierwszym sklasyfikowano bazę grafową Neo4j.

Redis

Redis jest aktualnie najpopularniejszą bazą danych typu klucz-wartość. Pod względem zgodności z aksjomatem CAP baza ta działa w modelu CP (spójność i tolerancja na partycjonowanie bez zapewnienia ciągłej dostępności danych). System został napisany przez Salvatore Sanfilippo w języku C i opublikowany w roku 2009. Aktualnie prace nad rozwojem bazy Redis są finansowane przez firmę Redis Labs i jest ona dostępna na licencji BSD[2] (Redis Labs, 2016). Redis jest magazynem struktur danych, które są przechowywane w pamięci operacyjnej komputera (Seguin, 2015). Jest on powszechnie używany jako baza danych, pamięć podręczna i broker wiadomości (Haber, 2017). Redis ma swój własny język zapytań. Ponadto biblioteki pozwalające na dostęp do bazy Redis są dostępne dla większości popularnych języków programowania (Macedo, Oliveira, 2011).

W tradycyjnych bazach wykorzystujących model „klucz-wartość", zarówno klucz jak i wartość jest ciągiem znaków (Carlson, 2013). W bazie Redis wartość nie jest natomiast ograniczona do prostego ciągu znaków, ale może również zawierać bardziej złożone struktury danych (RedisLabs, 2017). Lista dostępnych typów danych w bazie Redis obejmuje (Da Silva, Tavares, 2015):

- łańcuchy znaków (*string*), które mogą też być traktowane jako liczby (*integer*, *float*), teksty (nieformatowane, lub w formacie XML, JSON, HTML) lub jako ciągi binarne (pliki wideo, pliki graficzne oraz audio) w zależności jak są one wykorzystywane przez aplikacje; wartość pola typu String nie może przekroczyć 512 MB.
- listy, które mogą być traktowane jako proste kolekcje (*collection*), stosy (*stack*) lub kolejki (*queue*),
- tablice mieszające/tablice z haszowaniem (*hash table*), czyli indeksowane tablice pozwalające wyszukać wartość za pomocą indeksu – na podstawie podanego klucza funkcja mieszająca wyznacza indeks wartości w tablicy,
- zbiory (*set*) będące nieuporządkowaną kolekcją unikalnych wartości; wewnętrznie zbiór jest zaimplementowany za pomocą tablic z haszowaniem,
- zbiory sortowane (*sorted set*) są bardzo podobne do zbiorów, lecz każdej wartości jest przyporządkowana waga, według której zbiór jest posortowany,
- *HyperLogLog* będący specjalnym typem danych pozwalającym realizować algorytm HyperLogLog do określenia liczby unikalnych wartości w zbiorze; algorytm ten opiera się na probabilistycznym określeniu liczby elementów (Davis) i pozwala bardzo szybko określić liczbę elementów zbioru z wysokim prawdopodobieństwem, wykorzystując do tego bardzo małą ilość pamięci operacyjnej (Flajolet i in., 2007).

[2] Licencja BSD – (Berkeley Software Distribution License) licencja na dystrybucję oprogramowania zgodna z zasadami wolnego oprogramowania, opracowana na Uniwersytecie Kalifornijskim w Berkeley.

Wymieniony typ *sorted set* jest szczególnie istotny z punktu widzenia przetwarzania danych przestrzennych. Umożliwia uzyskanie uporządkowanego zbioru elementów, posortowanego według wag, gdzie wagą jest indeks geoprzestrzenny. W Redis zbiory sortowane w przypadku zastosowania ich do przechowywania danych przestrzennych są nazywane *geoset*. Każdy *geoset* składa się z jednego lub większej liczby elementów, z których każdy składa się z identyfikatora oraz pary współrzędnych. Podobnie jak w przypadku wszystkich struktur danych w Redis, dane geoprzestrzenne są przetwarzane za pomocą podzbioru prostych w użyciu poleceń (Da Silva, Tavares, 2015). W typie danych *geoset* wykorzystywany jest mechanizm indeksowania przestrzennego *geohashing*. Jest to rozwiązanie zaproponowane przez Gustavo Niemeyera w 2008 roku, pozwalające na wydajne indeksowanie danych przestrzennych. Celem idei indeksu *geohash* jest kodowanie współrzędnych punktów jako krótkich ciągów znaków, które mają być użyte w adresach internetowych. *Geohash* jest ciągiem binarnym, w którym każdy element wskazuje na położenie obiektu w poszczególnych sektorach coraz to mniejszych obszarów (Fox i in., 2013). Jest to hierarchiczna struktura, która dzieli przestrzeń na sektory o kształcie siatki. *Geohash* oferuje możliwość zapisu lokalizacji obiektu z dowolną precyzją. Usuwanie znaków z końca kodu powoduje stopniową utratę precyzji. Inną istotną cechą tego rozwiązania jest fakt, iż miejsca w bezpośrednim sąsiedztwie często przedstawiają podobne prefiksy – im dłuższy wspólny przedrostek, tym bliżej siebie są te miejsca. Dzięki takiemu rozwiązaniu poszukując elementów w bliskim sąsiedztwie nie ma potrzeby porównywania całego zestawu współrzędnych obiektów, lecz tylko najbardziej znaczące znaki *geohash*.

Redis umożliwia wykonanie podstawowych operacji na danych przestrzennych (Redis Labs, 2016). Umożliwia dodanie obiektu opisanego współrzędnymi (GEOADD), zwrócenie informacji o lokalizacji obiektu (GEOPOS), zwrócenie *geohash* obiektu (GEOHASH), zwrócenie listy obiektów w zadanej odległości od podanego punktu (GEORADIUS) lub od danego obiektu (GEORADIUSBYMEMBER). Poniżej przedstawiony przykłady: (1) zapisu współrzędnych, (2) odczytu współrzędnych oraz (3-5) wykonywania prostych operacji geoprzestrzennych.

1. Przykład zapisania za pomocą polecenia GEOADD współrzędnych kilku punktów w zbiorze typu *geoset* o nazwie *points* – wejście do Gmachu Głównego Politechniki Warszawskiej oraz brama główna pałacu Changdeokgung(창덕궁과) w Seulu
   ```
   GEOADD points 21.010792 52.220425 PW 126.990003 37.577783
   Changdeokgung
   ```
2. Przykład odczytania za pomocą polecenia GEOPOS współrzędnych zapisanych w elemencie o nazwie PW
   ```
   GEOPOS points PW
   1) 21.010792
   2) 52.220425
   ```
3. Przykład obliczenia odległości pomiędzy dwoma elementami zbioru *points*
   ```
   GEODIST points PW Changdeokgung
   "7742920.4626"
   ```
4. Przykład pobrania wartości *geohash* dla wybranych punktów
   ```
   GEOHASH points PW Changdeokgung
   1) "u3qcn47w4b0"
   2) "wydmc8z63w0"
   ```

5. Przykład wyboru wszystkich obiektów będących elementami zbioru *points* znajdujących się w odległości 50 km od punktu o współrzędnych 21°E 52°N z podaniem wyliczonej odległości:

```
GEORADIUS points 21 52 50 km WITHDIST
1) "PW"
2) "24.5282"
```

MongoDB

MongoDB jest aktualnie najpopularniejszą bazą danych typu magazyn dokumentów (*dokument store*). Pod względem zgodności z aksjomatem CAP system pracuje w modelu CP (spójność i tolerancja na partycjonowanie, bez zapewnienia ciągłej dostępności danych). Został opracowany przez firmę 10gen w 2007 roku. Aktualnie jest rozwijany przez firmę MongoDB Inc. i jest dostępny na licencji opartej na GNU AGPL[3] i Apache License[4]. MongoDB został napisany w językach C++ i C, a jako język zapytań wykorzystywany jest JavaScript. Baza pozwala na przechowywanie dokumentów w formacie JSON (*JavaScript Object Notation*). Dokumentom tym nadawane są unikalne identyfikatory. Do przechowywania danych geoprzestrzennych baza stosuje format GeoJSON, przy użyciu którego możliwe jest obsłużenie typów geometrycznych: punkt (*Point*), odcinek (*LineString*), łamana (*Polygon*), zbiór punktów (*MultiPoint*), zbiór odcinków (*MultiLineString*), zbiór łamanych (*MultiPolygon*) i zbiór elementów różnych typów (*GeometryCollection*). GeoJSON jest też wykorzystywany do konstruowania zapytań przestrzennych. Dokument w tym formacie zawiera pole o nazwie 'type' specyfikujące typ obiektu GeoJSON oraz pole o nazwie 'coordinates' zawierające współrzędne obiektu w formacie długość i szerokość geograficzna (w układzie WGS 84), z założeniem że długość geograficzna jest podawana jako pierwsza współrzędna. Poniżej przedstawiono 3 przykłady zapisu w formacie GeoJSON.

1. Przykład sposobu zapisu informacji o wybranym obiekcie punktowym

```
{
  type: „Point”,
  coordinates: [21.010792,52.220425]
}
```

2. Przykład sposobu zapisu odcinka

```
{
  type: „LineString”,
  coordinates: [[21.010792, 52.220425],[126.990003, 37.577783]]
}
```

3. Przykład sposobu zapisu łamanej zamkniętej

```
{
  type: „Polygon”,
  coordinates: [[[0, 0],[21.01, 52.22],[126.99, 37.58],[0, 0]]]
}
```

[3] Licencja GNU AGPL (GNU Affero General Public License) – licencja na użytkowanie oprogramowania uruchamianego po stronie serwera, gdy nie dochodzi w rzeczywistości do dystrybucji oprogramowania, co nie nakłada na użytkownika obowiązku udostępnienia kodu źródłowego aplikacji.

[4] Licencja Apache License – licencja na dystrybucję oprogramowania zgodna z zasadami wolnego oprogramowania, opracowana przez Apache Software Foundation.

Indeksowanie przestrzenne jest stosowane w bazach danych przestrzennych i ma na celu optymalizację zapytań przestrzennych. Baza MongoDB obsługuje dwa typy indeksów przestrzennych: indeks „2d" pozwalający indeksować lokalizacje obiektów na płaszczyźnie oraz indeks o nazwie „2dsphere", który umożliwia operacje na powierzchni sfery. Zapytania geoprzestrzenne dostępne w MongoDB to: zawieranie, przecięcie i sąsiedztwo. W MongoDB można wykonywać następujące operacje geoprzestrzenne:

- wybranie obiektów przecinających się z wybranym obiektem ($geoIntersects),
- wybranie obiektów zawartych w wybranym obiekcie ($geoWithin),
- wybranie obiektów w sąsiedztwie wybranego obiektu ($near, $nearSphere),
- utworzenie prostokątnego obszaru na podstawie podanych par współrzędnych punktów do wykonania analizy zawierania ($box),
- utworzenie kolistego obszaru na podstawie pary współrzędnych punktu centralnego do wykonania analizy zawierania ($center, $centerSphere),
- uzyskanie informacji o geometrii obiektu w formacie GeoJSON ($geometry),
- określenie maksymalnej odległości do ograniczenia liczby wyników zapytań o sąsiedztwo ($maxDistance),
- określenie minimalnej odległości do ograniczenia liczby wyników zapytań o sąsiedztwo ($minDistance).

Cassandra

Cassandra jest aktualnie najpopularniejszą nierelacyjną bazą danych typu NoSQL typu rodziny kolumn (*column family*, *wide column*). Zgodnie z aksjomatem CAP baza pracuje w modelu AP (dostępność i tolerancja na partycjonowanie, bez zapewnienia spójności danych). Projekt Cassandra był wzorowany na bazie Bigtable opracowanej przez Google oraz bazie Dynamo opracowanej przez Amazon. Baza została opracowana przez firmę Facebook w roku 2008. Całość kodu została napisana w języku Java. W kolejnych latach kod źródłowy został opublikowany na zasadach *open source* i przekazany do Apache Foundation. Baza jest dostępna na licencji Apache License 2.0. Odpowiednikiem tabeli w bazie Cassandra jest rodzina kolumn, w której kolumna jest najmniejszą jednostką zawierającą dane, składa się z klucza, wartości i pola typu *timestamp* (Amirian i in., 2013). Rodzina kolumn jest wewnętrznie przetwarzana jako sortowana mapa sortowanych map (Neeraj, 2015). Językiem zapytań jest CQL (*Cassandra Query Language*) wzorowany na języku SQL (Singh, 2013). Ponieważ Cassandra promuje denormalizację bazy danych, podczas pracy z nią użytkownik nie musi stosować zasad normalizacji danych (znanych z baz relacyjnych). W Cassandra schemat bazy danych został zaprojektowany tak, aby umożliwiać zapis i odczyt danych w sposób, w jaki są one przetwarzane w aplikacji (Padalia, 2015). Eliminuje to problem impedancji znany z baz relacyjnych. Lista dostępnych typów danych obejmuje (Yarabarla, 2017):

- ciągi znaków (*string*),
- liczby całkowite (*int*, *bigint*, *smallint*, *tinyint*, *varint*),
- liczby rzeczywiste (*float*, *double*, *decimal*),
- typ *timestamp* pozwalający przechowywać datę i czas,
- typ UUID (*Universally Unique Identifier*), będący bardzo dużą liczbą generowaną losowo za pomocą algorytmu zapewniającego jej globalną unikalność (ta sama wartość nie zostanie nigdy wygenerowana po raz drugi),

- typ logiczny (*boolean*),
- typ *Blob* (*binary large object*) służący do przechowywania nieustrukturalizowanych danych binarnych,
- kolekcje (*collection*), wśród których należy wyróżnić: listy (*list*) – nieunikalne, uporządkowane kolekcje zawierające jeden lub więcej elementów, zbiory (*set*) – kolekcje unikalnych wartości oraz mapy (*map*) – kolekcje par klucz–wartość,

Wśród wymienionych typów brak jest typu dedykowanego do przechowywania danych przestrzennych. Można do tego celu zastosować jednak kolekcje, ale Cassandra nie dostarcza żadnych mechanizmów wspomagających przetwarzanie tych danych. Każda aplikacja związana z GIS, która wymaga wczytywania dużych ilości danych i szybkiego wyszukiwania danych przy użyciu prostych kwerend, może efektywnie korzystać z baz danych typu rodzina kolumn (Amirian i in., 2013). Powstają zatem zewnętrzne rozwiązania rozszerzające możliwości bazy Cassandra o indeksowanie przestrzenne i mechanizmy przetwarzania danych geoprzestrzennych. Rozwiązania rozszerzające jej funkcjonalność opierają się na dodaniu indeksowania typu *geohash* znanego z bazy Redis, pozwalającego na kodowanie pary współrzędnych „długość, szerokość geograficzna". Indeks *geohash* jest przypisywany do danych przestrzennych podczas ich zapisu. Dodatkowo język CQL zostaje rozszerzony o możliwość pisania zapytań przestrzennych (Brahim i.in., 2016).

Neo4j

Baza danych Neo4j została opracowana przez Neo Technology, Inc. w 2007 roku. Kod źródłowy został napisany w języku Java. Wersja bezpłatna jest dostępna na licencji GPL[5], natomiast wersja komercyjna na licencji AGPL (Gupta, Neo4j Essentials, 2015). Językiem zapytań stosowanym w Neo4j jest język Cypher, którego składnia jest wzorowana na języku SQL (Kemper, 2015). W bazie danych występują dwa typy elementów: węzły, będące odpowiednikami obiektów oraz krawędzie łączące węzły, odzwierciedlające relacje zachodzące pomiędzy węzłami. Węzły mogą być proste, składające się jedynie z nazw lub złożone, zawierające atrybuty obiektów. Krawędzie maja nazwę identyfikującą typ połączenia oraz mogą posiadać kierunek. Przemieszczanie się po krawędziach pomiędzy węzłami nazywane jest trawersowaniem i pozwala na szybki dostęp do obiektów powiązanych ze sobą (Baton, Van Bruggen, 2017). Neo4j nie ma mechanizmów operacji na danych geoprzestrzennych, ale dostępne są one w Neo4j Spatial – bibliotece narzędzi dla Neo4j, która ułatwia udostępnianie danych przestrzennych (Neo4j Spatial, 2017). W szczególności można dodawać indeksy przestrzenne do już zlokalizowanych danych i wykonywać operacje przestrzenne na danych, takie jak wyszukiwanie danych w określonych regionach lub w określonej odległości od wskazanego punktu (Neo4j Spatial, 2017). Biblioteka udostępnia podstawowe typy geometryczne, takie jak: punkt (*Point*), odcinek (*LineString*), łamana (*Polygon*). Elementy przestrzenne są indeksowane za pomocą indeksu RTree. R-Tree to struktura danych wspomagająca wyszukiwanie obiektów w przestrzeni wielowymiarowej, w której do opisania obiektów wielowymiarowych wykorzystywane są minimalne regiony pokrywające – najmniejsze pudełka okalające obiekt. Podczas wyszukiwania danych przestrzennych dostępne są operacje topologiczne: *contains*, *within*, *intersects*, *covers*, *disjoint*.

[5] Licencja GPL (GNU General Public License) – licencja przekazująca użytkownikom wolność uruchamiania programu w dowolnym celu, wolność analizowania jak program działa i dostosowywania go do swoich potrzeb, wolność rozpowszechniania niezmodyfikowanej kopii programu oraz wolność udoskonalania programu i publicznego rozpowszechniania własnych ulepszeń.

Podsumowanie

Aktualnie dostępne mechanizmy przechowywania i przetwarzania danych przestrzennych w bazach NoSQL – w porównaniu do mechanizmów dostępnych w systemach baz relacyjnych takich jak: „Oracle Spatial and Graph"lub „PostGIS" – są znacznie uproszczone. Zaletami rozwiązań dostępnych w bazach relacyjnych są: stabilność rozwiązań (wieloletnia obecność na rynku, która doprowadziła do znacznych udoskonaleń), duża liczba dostępnych funkcji pozwalających na analizy przestrzenne (zwykle spełniających wymagania współpracujących z nimi systemów GIS) oraz ugruntowana pozycja na rynku, co przekłada się na popularność rozwiązania oraz dużą liczbę specjalistów posiadających doświadczenie w pracy z tymi systemami. Ograniczenia systemów opartych o relacyjne bazy danych wynikają raczej wyłącznie z ograniczeń samego modelu relacyjnego. Zastosowanie baz typu NoSQL eliminuje niektóre ograniczenia znane z baz relacyjnych: problemy skalowania i impedancji modeli danych. Zastosowanie typów danych zgodnych z typami danych przetwarzanych przez aplikacje pozwala uniknąć wielu problemów integracji i współbieżności.

Jeżeli przy wyborze konkretnego systemu do pracy z danych przestrzennymi przyjmiemy jako kryterium liczbę możliwych do wykonania analiz przestrzennych, oczywistym wyborem będą rozwiązania oparte na bazach relacyjnych. Jednak, gdy przewidywana jest bardzo duża ilość danych przestrzennych wymagająca zastosowania rozproszonej bazy danych, rozwiązanie oparte na bazach NoSQL mogą stać się korzystną alternatywą. Przy takim wyborze, należy się liczyć się z zaletami, ale również z ograniczeniami. Warto jednak mieć na uwadze, iż bazy NoSQL wciąż podlegają intensywnemu rozwojowi. Pojawiają się kolejne wersje wnoszące nowe możliwości. W związku z tym należy spodziewać się szybkiego rozwoju funkcjonalności związanej z przechowywaniem i przetwarzaniem danych przestrzennych w bazach NoSQL.

Podziękowania. Autor dziękuje dwóm anonimowym recenzentom za cenne wskazówki.

Finansowanie. Publikacja artykułu została sfinansowana ze środków Wydziału Geodezji i Kartografii Politechniki Warszawskiej przeznaczonych na badania dla doktorantów.

Literatura (References)

Amirian Pouria, Winstanley Adam C., Basiri Anahid, 2013: NoSQL storage and management of geospatial data with emphasis on serving geospatial data using standard geospatial web services. [In:] GIS Research UK (GISRUK), University of Liverpool.

Baton Jerome, Van Bruggen Rik, 2017: Learning Neo4j 3.x. Second Edition. Birmingham, Packt Publishing Limited.

Bradberry Russell, Lubow Eric, 2014: Practical Cassandra. Boston, MA, Addison-Wesley.

Brahim Mohamed Ben, Drira, Wassim, Filali, Fethi, Hamdi Noureddine, 2016: Spatial data extension for Cassandra NoSQL database. *Journal of Big Data*.

Carlson J.L., 2013: Redis in Action. Greenwich, Connecticut, United States, Manning Publications Co.

Carpenter Jeff, Hewitt Eben, 2016: Cassandra: The Definitive Guide. Sebastopol, CA, O'Reilly Media, Inc.

Celko Joe, 2014: Joe Celko's Complete Guide to NoSQL. Waltham, MA: Elsevier Inc.

Chinnachamy Arun, 2013: Instant Redis Optimization How-to. Birmingham, Packt Publishing Ltd.

Chinnachamy Arun, 2014: Redis Applied Design Patterns. Birmingham, Packt Publishing Ltd.

Chodorow Kristina, 2013: MongoDB: The Definitive Guide. Sebastopol, USA, O'Reilly Media, Inc.

Da Silva Maxwell, Tavares Hugo L., 2015: Redis Essentials. Birmingham, Packt Publishing Ltd.
Das Vinoo, 2015: Learning Redis. Birmingham, Packt Publishing Ltd.
Dasadia Cyrus, Nayak Amol, 2016: MongoDB Cookbook Second Edition. Birmingham, Packt Publishing Limited.
Davis K.J., n.d.: Redis HyperLogLog: Visualization and practical use with Node.js, Redis and Angular. Mountain View: RedisLabs.
Dzinko Rostyslav, 2015: Building Databases with Redis [Video]. Birmingham, Packt Publishing Ltd.
Edlich P.D., 2016: *NoSQL*. Retrieved from NoSQL Your Ultimate Guide to the Non-Relational Universe! 2016, 09 08. http://nosql-database.org/
Edward Shakuntala G., Sabharwal Navin, 2015: Practical MongoDB. New York, NY, Apress Media LLC.
Fauerbach Chris, 2017: Learning Neo4j Graphs and Cypher [Video]. Birmingham, Packt Publishing Limited.
Flajolet Philippe, Fusy Éric, Gandouet Olivier, Meunier Frédéric, 2007: HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. 2007 Conference on Analysis of Algorithms, AofA 07, Nancy, France, Discrete Mathematics and Theoretical Computer Science (DMTCS): 127-146.
Fox A., Eichelberger C., Hughes J., Lyon, S., 2013: Spatio-temporal Indexing in Non-relational Distributed Databases. Proceedings of the IEEE International Conference on Big Data.
França Wilson Docha, 2015: MongoDB Data Modeling. Birmingham, Packt Publishing Limited.
Ganyo Scott, 2014: Rapid Redis [Video]. Birmingham, Packt Publishing Ltd.
Ganyo Scott, 2015: Learning Redis [Video]. Birmingham, Packt Publishing Ltd.
Goel Ankur, 2015: Neo4j Cookbook. Birmingham, Packt Publishing Limited.
Gupta Sumit, 2015: Building Web Applications with Python and Neo4j. Birmingham, Packt Publishing Limited.
Gupta Sumit, 2015: Neo4j Essentials. Birmingham, Packt Publishing Limited.
Haber Itamar, 2017: Redis for Geospatial Data. Mountain View, RedisLabs.
Hills Ted, 2016: NoSQL and SQL Data Modelling. Basking Ridge, NJ, Technics Publications.
Hows David, Membrey Peter, Plugge Eelco, 2014: MongoDB Basics. New York, NY, Apress Media LLC.
Hows David, Membrey Peter, Plugge Eelco, Hawkins Tim, 2015: The Definitive Guide to MongoDB Third Edition. New York, NY, Apress Media LLC.
Jordan Gregory, 2014: Practical Neo4j. New York, NY, Apress Media LLC.
Kan C.Y., 2014: Cassandra Data Modeling and Analysis. Birmingham, Packt Publishing Limited.
Kemper Chris, 2015: Beginning Neo4j. New York, NY, Apress Media LLC .
Lal Mahesh, 2015: Neo4j Graph Data Modeling. Birmingham, Packt Publishing Limited.
Macedo Tiago, Oliveira Fred, 2011: Redis Cookbook. Sebastopol, O'Reilly Media, Inc.
McCreary Daniel G., Kelly Ann M., 2014: Making Sense of NoSQL. Shelter Island, New York, Manning Publications Co.
Nayak Amol, 2013: Instant MongoDB. Birmingham, Packt Publishing Limited.
Neeraj Nishant, 2015: Mastering Apache Cassandra – Second Edition. Birmingham, Packt Publishing Limited.
Nelson Jeremy, 2016: Mastering Redis. Birmingham, Packt Publishing Ltd.
Neo Technology, 2017: The Neo4j Developer Manual v3.1. Neo Technology.
Neo4j Spatial, 2017: Neo4j Spatial. Retrieved from Neo4j Spatial: (2017, 08 31). http://neo4j-contrib.github.io/spatial/
O'Higgins Niall, 2011: MongoDB and Python. Sebastopol, CA, O'Reilly Media, Inc.
Padalia Nitin, 2015: Apache Cassandra Essentials. Birmingham, Packt Publishing Limited.
Palmer Matt, 2013: Instant Redis Persistence. Birmingham, Packt Publishing Ltd.
Raj Sonal, 2015: Neo4j High Performance. Birmingham, Packt Publishing Limited.
Redis Labs, 2016: Redis. Retrieved from Redis: (2016.09.08). http://redis.io/
RedisLabs, 2017: An introduction to Redis data types and abstractions. Retrieved from Redis: (2017, 07 18). https://redis.io/topics/data-types-intro
Robinson Ian, Webber Jim, Eifrem Emil, 2015: Graph Databases. Sebastopol, CA, O'Reilly Media, Inc.
Sadalage Pramod J., Fowler Martin, 2012: NoSQL Distilled. Boston, Addison-Wesley Professional.

Seguin Karl, 2015: The Little Redis Book. Licensed under the Attribution-NonCommercial 3.0 Unported license.
Sharma Sanjay, 2014: Cassandra Design Patterns. Birmingham, Packt Publishing Limited.
Singh Amresh, 2013: Instant Cassandra Query Language. Birmingham, Packt Publishing Limited.
Strauch Christof, n.d.: NoSQL Databases. Stuttgart, Stuttgart Media University.
Tiwari Shashank 2011: Professional NoSQL. Indianapolis, John Wiley & Sons, Inc.
Vaish Gaurav, 2013: Getting Started with NoSQL. Birmingham, Packt Publishing Limited.
Van Bruggen Rik, 2014: Learning Neo4j. Birmingham, Packt Publishing Limited.
Vohra Deepak, 2015: NoSQL Web Development with Apache Cassandra. Cengage Learning PTR.
Vohra Deepak, 2015: Pro MongoDB Development. New York, NY, Apress Media LLC.
Vukotic Aleksa, Watt Nicki, Abedrabbo Tareq, Fox Dominic, Partner Jonas, 2015: Neo4j in Action. Shelter Island, NY, Manning Publications Co.
Yarabarla Sandeep, 2017: Learning Apache Cassandra (Second Edition). Birmingham, Packt Publishing Limited.

Streszczenie

Relacyjne bazy danych mają ugruntowaną podstawę koncepcyjną i nadal silną pozycję na rynku. Warto jednak zauważyć, że leżąca u ich podstaw koncepcja powstała na początku lat 70. z uwzględnieniem możliwości dostępnych wtedy komputerów. Dzisiejsze komputery osobiste mają parametry techniczne pozwalające przechowywać znacznie większe ilości danych i przetwarzać je w nieporównywalnie szybszy sposób. Mimo ewolucji systemów zarządzania relacyjnymi bazami danych i dostosowywania ich do coraz to nowych potrzeb, pojawiają się alternatywne rozwiązania o innych cechach i możliwościach, pozwalające miedzy innymi na przechowywanie i przetwarzanie jeszcze większej ilości danych. Ich rozwój wynika częściowo z potrzeby eliminacji ograniczeń baz relacyjnych, a częściowo z coraz większych możliwości dostępnego sprzętu komputerowego. Tego typu rozwiązaniem są bazy określane jako bazy NoSQL. Są to rozwiązania klasy open source, oparte na nierelacyjnym modelu danych i dostosowane do działania w środowisku rozproszonym. Ich popularność rośnie na tyle szybko, że trzeba zwrócić uwagę na możliwość ich zastosowania do zarządzania i analizowania danych przestrzennych. Celem niniejszego artykułu jest przedstawienie podstawowych zalet i wad baz NoSQL w kontekście możliwości operowania na danych przestrzennych. Dla projektantów i użytkowników systemów GIS bazujących na popularnych obecnie platformach takich jak: „Oracle Spatial and Graph" bądź „PostGIS" podstawowymi pojęciami związanymi z zarządzaniem bazami danych są: typy geometryczne danych, indeksowanie przestrzenne, operatory przestrzenne, funkcje i procedury realizujące analizy przestrzenne, modele topologiczne i sieciowe. Podstawowe pojęcia stosowane w NoSQL są znacząco inne. Artykuł ma charakter przeglądu literaturowego z minimalną liczbą przykładów mających na celu jedynie zasygnalizowanie sposobu pracy baz typu NoSQL.

Abstract

Relational databases have a well-established conceptual foundation and are still holding a strong market position. It is worth noting, however, that the underlying concept was created in the early 1970s, taking into account parameters of then-available computers. Today's personal computers have technical features that allow to store much larger amounts of data and process them in an incomparably faster way. Despite the evolution of relational database management systems, and adaptation them to ever-changing needs, alternative solutions have emerged that offer different features and capabilities for storing and processing even more data. Their development is due in part to the need to eliminate relational database constraints, and in part to the increasing availability of computer hardware. This type of solution is called the NoSQL database. They are open source solutions based on a non-relational data model and adaptation to a distributed environment. Their popularity grows fast enough so that attention should be paid to their usefulness for management and analysis of spatial data. The objective of this paper is to present the main advantages and disadvantages of NoSQL

databases in the context of ability to manipulate spatial data. For designers and users of GIS systems based on popular platforms such as „Oracle Spatial and Graph" and „PostGIS", basic concepts related to database management are: geometric types of data, spatial indexes, spatial operators, spatial functions and procedures, topological and network models. The basic concepts used in NoSQL are significantly different. The paper presents the literature review with a minimal number of examples aiming only at signalling the NoSQL database operations

Dane autora / Autor details:

mgr inż. Michał Wyszomirski
https://orcid.org/0000-0002-5407-0536
michal.wyszomirski@gmail.com

Przesłano / Received 3.10.2017
Zaakceptowano / Accepted 18.12.2017
Opublikowano / Published 15.02.2018