



TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

**DOKUMENTUM ALAPÚ ADATBÁZISOK
VIZSGÁLATA TEJLESÍTMÉNY ÉS
TERHELHETŐSÉG SZEMPONTJÁBÓL**

Szerző:

Nagy András

mérnök informatikus BSc. szak, III. évf.

Konzulens:

Kertész Gábor

műszaki tanár

Tartalomjegyzék

1. Bevezetés.....	1
2. Adatbázis-kezelő rendszerek csoportosítása	2
2.1. NoSQL típusú adatbázisok csoportosítása	2
2.1.1. Kulcs/érték tárolók (Key/Value Stores).....	2
2.1.2. Gráf adatbázisok	3
2.1.3. ColumnStores vagy BigTable adatbázisok.....	3
2.1.4. Dokumentum-alapú tárolók	4
3. Adattárolás új szemszögből	5
3.1. A NoSQL történelmi áttekintése	6
3.2. Dokumentum-alapú tárolók összehasonlítása az RDBMS rendszerekkel.....	7
3.2.1. ACID, CAP tétel	8
4. MongoDB bemutatása	10
4.1. A MongoDB történelmi áttekintése	10
4.2. A dokumentumok bemutatása.....	10
4.2.1. ObjectId.....	12
4.2.2. Indexelés	12
4.2.3. CRUD a MongoDB-ben	13
4.3. Skálázhatóság a MongoDB-ben.....	16
4.4. Replikáció a MongoDB-ben	17
4.5. MongoDB tárolási módja, technikai korlátai.....	19
5. Összehasonlító mérési eredmények.....	20
5.1. Felhasznált eszközök	20
5.2. Adatbázisaim.....	20
5.3. A PHP Webalkalmazás ismertetése	21
5.4. Mérési műveletek.....	23

5.5. A mérések eredményei	24
5.6. Mérés Apache Benchmark felhasználásával.....	26
6. Konklúzió	30
7. Összefoglalás	32
8. Ábrajegyzék	33
9. Irodalomjegyzék	34

Bevezetés

A Webes világban az egyik legfontosabb igény, hogy egy oldal minimális idő alatt kerüljön betöltésre. Egy Webes alkalmazás válaszidejének legnagyobb részét az adatbázis műveletek eredményére való várakozás teszi ki. A helyesen megválasztott adatbáziskezelő-rendszer használata lehet az alapja egy kiemelkedő teljesítményű alkalmazásnak. Ehhez a döntéshez mérlegelnünk kell a rendelkezésre álló erőforrásokat, a leendő adatok mennyiségét, fontos megbecsülni a hálózati adatforgalmat, érdemes megtervezni milyen adatbázis műveleteket szeretnénk használni, illetve mennyire fontos, hogy strukturált formában legyenek az adatok tárolva. Ezen tervezési szempontok alapján érdemes eldönteni, milyen adatbáziskezelő-rendszert használjunk.

A relációs adatmodellt használó modelleken kívül egyre több, más típusú rendszer jelenik meg, amelyek az úgynevezett NoSQL adatbázisok csoportjába tartoznak. Ebbe a csoportba jellemzően olyan adatbázis kezelők sorolhatók, amelyek nem használják az SQL nyelvet, illetve a relációs adatmodellt sem követik. Az egységes SQL nyelv helyett minden adatbázis kezelő különböző lekérdezési nyelvet használ, illetve többféle adatmodellt is használnak. A legtöbb NoSQL rendszer működése a teljesítményre optimalizált, amely hatására az adat integritásával kapcsolatos ACID szabályt is megszegve, olyan módszerek sem kerültek bele, mint például a már ismert tranzakciókezelés.

Kutatásom során bemutatásra kerül a legnépszerűbb NoSQL adatbáziskezelő-rendszer a MongoDB, különös figyelmet fordítva a rendszer sajátosságaira, különbségeire a relációs adatmodellhez képest. Jellegzetességeinek ismertetése után összehasonlításra kerül az egyik legnépszerűbb adatbázis kezelő rendszerrel a MySQL-el. Az összehasonlítás fő szempontja a műveletek végrehajtási ideje írási műveletek vagy olvasási lekérdezések esetén. Megvizsgálásra kerülnek, különböző méretű adatbázisok használatakor a lekérdezési idők, egy saját PHP nyelven írt Webalkalmazás segítségével. Méréseim részét képezi a konkurens lekérdezések vizsgálata is. Munkám zárásaként mérési eredményeimet felhasználva megállapításra kerülnek, melyek azok az esetek, amikor érdemes használni a bemutatásra került MongoDB-t, és mely esetekben érdemes továbbra is a relációs modellt alkalmazni.

1. Adatbázis-kezelő rendszerek csoportosítása

Jelenlegi ismereteink szerint az adatbázisokat csoportosíthatjuk olyan módon, hogy relációs (SQL), vagy nem relációs (NoSQL) adatbázisról van-e szó.

Dolgozatom során többször a relációs adatmodell helyett a klasszikus adatmodell szókapcsolatot fogom használni.

Relációs adatbázisról akkor beszélünk, ha az adatok egy táblázatban, sorokban és oszlopokban vannak eltárolva, azaz egy előre definiált struktúra alapján építjük fel az adatbázisunkat. Jelenleg a világ legnépszerűbb adatbáziskezelő-rendszerei [1] zömével idetartoznak.

Az SQL nyelv megjelenése körülbelül a 70-es évek elejére tehető, egy időben a relációs adatmodell kialakulásával. A relációs adatbázisok jellemzője, hogy hasonló lekérdezési nyelvet használnak a különböző rendszerek.

A legnépszerűbb relációs adatbázis-kezelő rendszerek (RDBMS): Oracle SQL, MySQL, MSSQL, PostgreSQL, SQLite.

A nem relációs adatbázisok egy tágabb csoportot alkotnak. A NoSQL jelentése lehet „Not only SQL”, vagy más értelmezésben, „no SQL”. Ez azt jelenti, hogy ebben a csoportban olyan rendszerek vannak, amelyek nem használnak SQL nyelvet a műveletek lekérdezéséhez, illetve nem relációs adatmodellt használnak.

1.1. NoSQL típusú adatbázisok csoportosítása

A NoSQL adatbázisokat kisebb csoportokra is bonthatjuk, adattárolási módjaik alapján.

Négy kiemelten fontos csoportot érdemes kicsit alaposabban is megismerni. [2] Érdemes tudni, hogy folyamatosan jelennek meg újabb és újabb csoportok, megoldások, ezek azonban a piac méretéhez viszonyítva még kis felhasználói körrel rendelkeznek.

1.1.1. Kulcs/érték tárolók (Key/Value Stores)

Ezen tárolási mód fő jellemzője az egyszerűség, ami magába foglalja a séma strukturátlanságát. Ez annyit jelent, hogy különböző sorokban akár teljesen különböző típusú adatokat tárolhatunk, ellentétben a relációs adatmodellel, ahol az adattábla

meghatározza az adatbázis szerkezetét. Az adatbázisunkban a kulcs alapján hajthatjuk végre a szűréseket, kereséseket.

Lekérdezéseket csak kulcs alapján tudunk végrehajtani, illetve csak alapvető műveleteket hajthatunk végre az adatbázisunkon. Nem támogatja a triggerek, megszorítások, procedúrák használatát, ezért bonyolultabb adatbázis műveleteket nem érdemes, vagy szinte lehetetlen megvalósítani.

Néhány ilyen elven működő adatbázis például a DynamoDB, AzureTable Storage, Riak, Redis, Oracle NoSQL Database, MemcacheDB.

1.1.2. Gráf adatbázisok

Az ilyen típusú adatbázis az adatokat gráf elrendezésben tárolja. Ehhez a tároláshoz éleket, csomópontokat és tulajdonságokat használ fel. Minden egyes csomópont tartalmaz egy mutatót, ami a szomszédos elemeket definiálja.

Manapság egyre nagyobb szerepet kap ez a tárolási mód, különösen a Webes világban. Kiválóan használhatóak összetett hálózatok vizsgálatára is ezek a megoldások.

Ilyen rendszerek például a Neo4J, InfiniteGraph, DEX, Titan, Trinity.

1.1.3. Column Stores vagy BigTable adatbázisok

Sok adat tárolására alkalmas, illetve oszlop szerint végzi a keresést, tárolást. Mint ismeretes az SQL adatbázisok soronként tárolják az adatokat. Minden sor fizikailag is külön helyezkedik el. Egy-egy mező keresése úgy történik, hogy először megkeressük az aktuális sort, majd kiválasztjuk a sorban levő mezőt, amit kerestünk. Ezzel ellentétben ez a módszer oszlopok szerint tárolja el a szükséges információt.

Olyan esetekben érdemes használni, amikor egy jelentős méretű adatbázisból ki szeretnénk olvasni egyetlen oszlop értékeit. Például ha egy naplózó rendszerünk van, akkor nagyon gyorsan kilistázhathatjuk, hogy kik látogattak meg egy adott időintervallumban egy adott weboldalt.

A legismertebb ezen elven működő adatbázisok: ApacheHbase, Cassandra, BigTable.

1.1.4. Dokumentum-alapú tárolók

Nem tartalmaz klasszikus értelemben vett rekordokat, hanem egy sort egy dokumentum helyettesít. Egy dokumentum több típusú lehet, amit az adatbázis rendszere határoz meg. A teljesség igénye nélkül egy adatbázis tárolhatja az adatokat JSON, XML, BSON, PDF vagy akár egy Word fájlban is. Azért előnyös, hogy dokumentumokban tároljuk az adatokat, mert a felhasználó számára is könnyen olvasható az adatbázis. Szerkezetét tekintve leginkább a kulcs/érték tárolókhoz hasonlít, mivel egy dokumentumban is kulcs/érték párokat adhatunk meg, de különbség, hogy egyszerre akár több értékpárt is megadhatunk, illetve egy kulcsnak lehet több értéke is. [3] Továbbá ez a csoport hasonlít leginkább a relációs adatbázisokhoz is. A fő strukturális különbség a két rendszer között, hogy a dokumentum alapú tárolók rugalmas sémával rendelkeznek.

Főbb dokumentum-alapú tárolók: CouchDB, OrientDB, MongoDB.

Mivel a kutatásom alapjául használt MongoDB adatbázis-kezelő rendszer is ebbe a csoportba tartozik, ezért a későbbiek folyamán részletesebben is kitérek a dokumentum-alapú tárolókra, illetve a relációs adatbázis közötti különbségekre.

2. Adattárolás új szemszögből

Amikor megjelentek az első NoSQL rendszerek, a kérdés, ami megfogalmazódhatott az emberekben, hogy miért van szükség egy teljesen új modellre, adatbáziskezelő-rendszerre, ha a relációs adatbázisok szinte egyeduralkodóként vannak a piacon, minden fejlesztő ezeket ismeri, és használja.

Minél tovább tart egy lekérdezés, annál tovább terheli az adatbázisszervert, így több erőforrást vesz igénybe, aminek viszont nagyon fontos szerepe van a költségekben. Ezért érdemes olyan adatbáziskezelő-rendszert alkalmazni, amely a lehető legrövidebb végrehajtási idő alatt hajtja végre a műveletet. Ha sok felhasználó egyszerre hajtja végre a lekérdezéseket, akkor előbb-utóbb nőni fognak a végrehajtási idők, nem is beszélve arról, hogy a táblák mérete folyamatosan gyarapodik, és például egy SELECT utasításnál hatványozottan nőni fog a művelethez szükséges idő. A mérési eredményeimről a dolgozat későbbi részében fogok beszámolni. Ha hosszabbodnak a műveleti idők, akkor érdemes növelni az adatbázis kiszolgálónak vagy kiszolgálóknak a teljesítményét, amit elérhetünk a szervergép bővítéssel, vagy terhelés-elosztást vezetünk be, ami az SQL esetében nem is olyan egyszerű, illetve mindkét megoldás költségekkel jár. További nagy probléma a relációs adatbázisoknál a kötöttség. Mint tudjuk, nagy történelmi múltra tekint vissza, ami ebben az esetben hátrányt jelent vetélytársaival szemben. 70-es évek óta az informatika jelentős mértékben megváltozott, a tárolandó adatok mennyisége többszörösére nőtt, illetve a tárolandó adatok típusa is nagymértékben változott, strukturálatlan adatok halmaza tölti meg a Webes világot. Ezen adatok tárolására nem nyújt hatékony megoldást egy relációs adatmodellt követő adatbázis. Ilyen problémába ütközhetünk, ha egy közösségi portálon a közös ismerősöket szeretnénk megnézni. Ezt a klasszikus adatmodellel nehéz megvalósítani, joinok sorozatával tehetnénk erre kísérletet. Egyszerűbb megvalósítást biztosít a gráfos adatmodell. Egy másik példában a dokumentum-alapú adatbázis kezelés kínál hatékony megoldást. Blogok, hírek különböző cikkek esetében nagy népszerűségnek örvend a címkézés. Ez azt jelenti, hogy hozzárendelünk egy blogbejegyzéshez különböző kulcsszavakat, amelyek alapján végrehajthatunk különböző kereséseket, így a felhasználó könnyen hasonló témájú cikket tud olvasni igénye szerint. Ezt rekordokban azért nehéz megvalósítani, mert előreláthatóan nem tudjuk pontosan, hogy mennyi kulcsszót szeretnénk definiálni, illetve egy cikkhez több címke is tartozhat. Ezt a problémát a dokumentum-alapú adatmodell úgy valósítja meg, hogy egy attribútumhoz akár több érték is tartozhat.

Ezen fő alapgondolatokat figyelembe véve valósult meg a NoSQL világa. Tegyük meg egy rövid történelmi áttekintést!

2.1. A NoSQL történelmi áttekintése

1998-ban Carlo Strozzi használta először a NoSQL kifejezést. [4] Érdekesség, hogy Strozzi ezt a szót egy relációs adatbázis fejlesztése közben említette, azonban már nem használta az SQL nyelvet, miközben megtartotta a relációs adatmodellt. A mozaikszó feledésbe merült, egészen 2009-ig, amikor is Eric Evans újra a köztudatba hozta a kifejezést, ami egyfajta gyűjtőfogalomként szolgált a hagyományos adatmodellektől eltérő adatbázisoktól.

Előzmények már az 1960-as évek közepéig terjedtek, hiszen folyamatosan próbálkoztak a fejlesztők a klasszikus adatmodelltől eltérő struktúrákat létrehozni. Ekkor jelent meg a Multi Value adatbázis, amit a PICK operációs rendszerhez fejlesztettek. 1966-ban az IBM kifejlesztette a hierarchikus felépítésű IMS adatbázist, ami az Apollo űrprogram keretein belül valósulhatott meg. Később ez az adatbázis adta a szintén IBM által fejlesztett DB2-es rendszer alapját.

A következő előrelépés 1979-ben történt. Az AT&T létrehozta a dbm nevű adatbázisát, ami már tartalmazott például elsődleges kulcsot és hashelést. A 80-as években több utóda is megjelent a dbm-nek, például a Tdbm, Ndbm, Sdbm, és a dbm nyílt forráskódú rendszer a Gdbm. A Berkeley egyetem is létrehozott egy saját adatbázismotort 1994-ben.

Az említett adatbázisok nagy része ma már nem használatos, vagy nem rendelkeznek nagy jelentőséggel, ezzel elveszítve a párharcot a hagyományos szerkezetű adatbázisokkal szemben.

1997-ben megjelent első dokumentum alapú rendszer, ami a Metakit nevet kapta. 2005-ig fejlesztették, majd sikertelensége miatt abbahagyták a terméktámogatást.

Azonban az ezredforduló környékén elindult az a folyamat, hogy egyre több fejlesztőcsoport elkezdte a saját NoSQL adatbázisát létrehozni. Ezen rendszerek egy részét viszont már a mai világban is használjuk.

2000-ben jelent meg egy gráf adatbázis a Neo4j, 2005-ben pedig az Infogrid. 2000-ben objektum alapú adatbázis is jelent meg a piacon, például a db4o. 2005-ben jelent meg a CouchDB, amit három év múlva az Apache felvásárolt és jelenleg is fejleszti.

Ezen rendszerek voltak a NoSQL „forradalom” előhírnökei, azonban az igazi előrelépés 2004-től kezdődött.

A Google is elkezdte saját NoSQL adatbázisát fejleszteni. A keresőóriás volt az első világszinten elismert cég, aki nem kevés tőkét és energiát kezdett el befektetni ebbe a projektbe, ami a BigTable nevet kapta. [5] Az első két évben cikkek formájában jelentek meg az elképzelések és az elméleti háttér az új adattárolási módszerről. 2005-ben jelent meg a Google File System bemutatása, majd a BigTable, később pedig a MapReduceről szóló írások. A rendszer elsődleges célja, hogy rendkívüli mennyiségű adatot lehessen tárolni elosztott rendszerben, ami a MapReduce algoritmust használja. Mára a Google sok szolgáltatásában BigTable alapú rendszerekben tárolja információit, ilyen a GoogleMaps, a GoogleEarth, a Youtube vagy akár a Gmail. A Google által fejlesztett adatbázis nem nyílt forráskódú, ellentétben a legtöbb NoSQL típusú adatbázissal szemben. Fejlesztés közben azonban többször is jelentek meg dokumentációk, amelyek az adatbázis működését, felépítését bemutatták. Több rendszer fejlesztésének az alapját is ezek a publikációk befolyásolták, ilyen a Cassandra, HBase és a LevelDB.

2007-ben az Amazon Dynamo, 2009-ben pedig a Redis, a Riak, HBase és nem utolsósorban a MongoDB jelent meg.

Első sikerek között található a Facebook, ahol a Cassandra nevű kulcs/érték és oszlopos adatmodellű rendszere tárolta az üzeneteket 2010-ig. Ekkor a közösségi oldal a HBase-t választotta, amit a mai napig használnak. A jelenleg Apache által fejlesztett Cassandra sem merült feledésbe, ma az Instagram és a Twitter is használja.

2.2. Dokumentum-alapú tárolók összehasonlítása az RDBMS rendszerekkel

A dokumentum-alapú adatbázisokra jellemző a nyílt forráskódú licenz. Ezen csoportba tartozó rendszerek fejlesztése során különösen fontos szempont a lehető legkisebb végrehajtási idő, illetve a nagyfokú skálázhatóság. A skálázhatóság azt jelenti, hogy a kiszolgáló szerver illetve szerverek teljesítményének növelése a számítógép architektúrájának módosításával miképp valósítható meg. Kétféle skálázhatóságot ismerünk, ezek a vertikális, illetve a horizontális skálázhatóság. A vertikális skálázhatóságot a relációs adatmodellt használó adatbázis kezelők használják ki

hatékonyan. Ezen skálázhatósági eljárás megvalósítása kiszolgáló fizikai paramétereinek a bővítése, ami lehet memória, processzor vagy háttértároló.

A horizontális skálázhatóság jellemzője, hogy a teljesítmény növelése érdekében egy szerver helyett több különböző eszköz fogja kiszolgálni a lekérdezéseket. Ezen számítógépek fizikailag akár különböző helyen is elhelyezkedhetnek, azonban logikailag egy adatbázist szolgálnak ki. A legtöbb ismert adatbázis kezelő támogatja a horizontális skálázhatóságot, azonban relációs adatbázisok esetén kevésbé optimalizált, mint a dokumentum-alapú adatbázisok esetén.

2.2.1. ACID, CAP tétel

Az ACID tétel négy fő komponensből áll, amelyek teljesülése során az adatbázis integritása megvalósul. Az első elem az atomicitás. Ez azért felelős, hogy az adatbázison végrehajtott műveleteknek vagy mindegyike végrehajtsódjon, vagy egyik sem. A konzisztencia azt szolgálja, hogy konzisztens állapotból az adatbázis műveletek után is konzisztens állapotba kerüljön. Az izoláció abban kap fontos szerepet, hogy az egyes tranzakciókat elkülönítse egymástól, azaz úgy tűnjön egy felhasználónak, hogy csak ő dolgozik az adatokon. Az utolsó fontos kritérium, ami nélkül az integritás nem jöhetne létre a tartósság. Ez a feltétele, hogy végrehajtott tranzakciókat helyre lehessen állítani például egy rendszerhiba esetén.

A tétel teljesülése a feltétele annak, hogy az adatbázisunkon tranzakció műveletet végezhesünk. A leggyakrabban használt relációs adatbázisok, például a MySQL használják a tranzakció kezelést, azonban a dokumentum-alapú adatbázisok nem támogatják a tranzakciókat. Ennek fő oka teljesítményorientáltság. A tranzakciók sikeres elvégzéséhez nélkülözhetetlen nagyszámú commit utasítás, ami negatívan hat a végrehajtási idők nagyságára.

A CAP tétel az elosztott rendszerekre vonatkozik, függetlenül az adattárolás módjától. A CAP egy mozaikszó, ami a consistency, availability és a partition tolerance angol szavak kezdőbetűjéből állnak. [6] A konzisztencia, hasonló jelentéssel bír, mint az ACID esetében. Az availability azaz a rendelkezésre állás azt mutatja meg, hogy a szerver mennyi ideig tudja kiszolgálni a klienst valamilyen komplikáció bekövetkezése nélkül, azaz a rendszer hibatűrését mutatja. Ezt jellemzően százalékkal szokták megadni, ami minél nagyobb annál megbízhatóbb az eszköz. A partition tolerance azzal a kérdéssel

foglalkozik, hogy mi történjen, ha az elsődleges partíció meghibásodik. Ekkor egy másodlagos partíció kerülhet a helyére, vagy az aktuális művelet nem lesz végrehajtva.

A tételt Eric Brewer fogalmazta meg, aki azt mondta, hogy terhelés-elosztott rendszerekben a három attribútumból egyszerre csak kettő teljesülhet. A Mongo a három paraméterből a rendelkezésre állást valósítja meg a legkevésbé.

Említésre méltó különbség van egy dokumentum-alapú és egy hagyományos SQL adatbázis tervezésében is. A relációs modell esetén egy több táblát tartalmazó adatbázisban az adattáblák kapcsolatát join műveletekkel valósíthattuk meg. Dokumentum-alapú világban is megvalósíthatjuk ezt az adatbázis tervezési módot a joinokhoz hasonló működésű, referenciák segítségével. [7] Másik lehetőség a beágyazott dokumentumok használata. Ebben az esetben egy collectionben tároljuk az összes adatot, ellentétbe az előző megoldással ahol annyi collection kellett a megvalósításhoz, amennyi adattáblánk eredetileg volt. Ilyen esetben azonban megjelenik a redundancia, illetve megszűnik a relációs adatmodellből ismert normalizált adatbázis. Minél mélyebb a dokumentum beágyazottsága, annál nagyobb a redundancia.

3. MongoDB bemutatása

3.1. A MongoDB történeti áttekintése

A MongoDB fejlesztése 2007-ben kezdődött a 10Gen nevű szoftverfejlesztő cég által C++ nyelven. [8] A New York-i székhelyű vállalat szintén 2007-ben alakult, ez volt a cég első és máig egyetlen nagy projektje, ugyanis mára már a MongoDB Inc. nevet viseli. Három alapítójuk Kevin P. Ryan, Dwight Merriman és Eliot Horowitz voltak, akik a Google egyik leányvállalatánál dolgoztak a DoubleClicknél. Szerencsájukra, nem kellett küzdeniük anyagi nehézségekkel, ugyanis körülbelül 81 millió dollárt sikerült befektetőktől kapniuk. A befektetők névsorában megemlíthető például az Intel, a Red Hat és a Sequoia Capital.

Az első verzió 2009-ben jelent meg, azóta töretlen a sikere a dokumentum-alapú adatbázisnak. A project nyílt forráskódú, ennek is köszönheti gyors fejlődését. 2012-ben az akkor még 10Gen néven került a The Wall Street Journal újság The Next Big Thing 2012-es listájában a kilencedik helyet érte el. Fejlesztési stratégiájuk kicsit rendhagyó volt, mivel a korai verziókban csak néhány funkciót lehetett elérni, és a későbbi verziók bővültek ki olyan eszközökkel, mint például a replikáció. Ennek a stratégiának az volt a fő célja, hogy rövid időn belül minél több emberhez elérjen az alap technológia, és a később fejlesztett funkciók kevésbé voltak szükségesek a legtöbb felhasználónak.

Kezdeti céljuk a fejlesztőknek, hogy egy olyan rendszert hozzanak létre, amely kiküszöböli a MySQL gyengeségeit, ma már azonban vannak, akik az új MySQL-t látják a MongoDB-ben.

3.2. A dokumentumok bemutatása

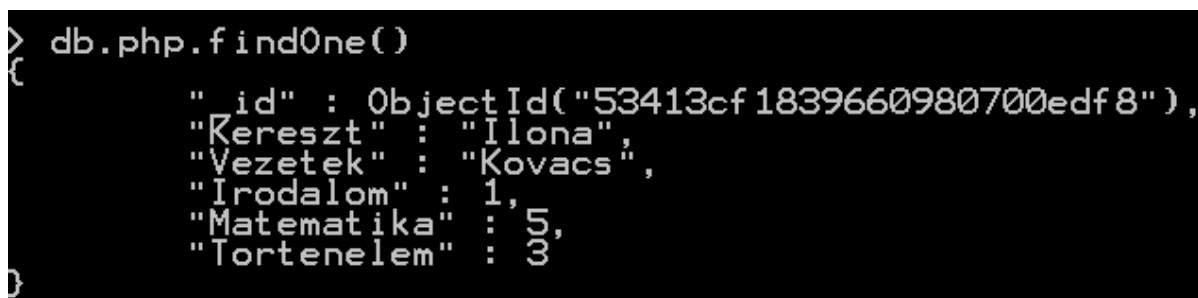
A MongoDB és egy SQL adattárolónak is a legnagyobb egysége az adatbázis. A klasszikus adatstruktúra alapján egy adatbázisban tábla vagy táblák helyezkednek el. [9] A MongoDB-ben strukturálatlan adatok vannak, ezért nem használható a tábla adattípus, helyette az úgynevezett collectiont vezették be. Szerepe hasonlít a táblához. Mint ismeretes a táblák rekordokból épülnek fel, ennek a MongoDB-ben a dokumentum felel meg.

A dokumentumokat a JavaScriptben először használt JSON formátumban adhatjuk meg. A JSON egy adatleíró nyelv, amely biztosítja az adatcserét szerver és kliens között. A formátum támogatja a strukturálatlan felépítés tárolását és megjelenítését.

MongoDB lekérdezési nyelve gyengén típusos, amely a string, integer, boolean, double és ObjectId adattípusokat használja. Az érdekesség, hogy a JSON támogatja a tömböket is, azaz egy kulcsnak akár adhatunk több értéket is, sőt akár egymásba is ágyazhatunk dokumentumokat. További előnye a JSON formátumnak, hogy a Weben egyre nagyobb népszerűségnek örvendő XML adatleíró nyelvvel teljes mértékben kompatibilis, illetve mára nyelv független lett. Hátránya, hogy nem teljes egészében támogatja az UTF-8 használatát, ezért érdemes ékezet nélküli karaktereket használni.

A Mongo JSON formátumhoz hasonlóan tárolja el a dokumentumokat, úgynevezett BSON típusban. A mozaikszó a Binary JSON rövidítést jelöli, és a fent említett jellemzők ugyanúgy igazak erre a formátumra is. [10] A BSON kialakítása során három fontos jellemzőt tartottak szem előtt: strukturálatlan, hordozható és hatékony. Az utóbbi azt jelenti, hogy az adat konvertálása BSON formátummá gyors legyen. Jellemző a BSON-ra, hogy a hatékonyság előnyére az olvashatóság, értelmezhetőség csökkent, illetve több extra információval is szolgál, például az indexelés, ami növeli a tárolási méretet.

Az 1. ábrán egy JSON formátumú dokumentumot láthatunk a MongoDB adatbázisunkban.



```
> db.php.findOne()  
{  
  "_id" : ObjectId("53413cf1839660980700edf8"),  
  "Kereszt" : "Ilona",  
  "Vezetek" : "Kovacs",  
  "Irodalom" : 1,  
  "Matematika" : 5,  
  "Tortenelem" : 3  
}
```

1. ábra: MongoDB egy dokumentuma JSON formátumban

BSON-ban a fájl tárolása a következőképpen történik:

A {„hello” : „world”} dokumentumot a MongoDB a következőképpen tárolja: „\x16\x00\x00\x00\x02hello\x00\x06\x00\x00\x00world\x00\x00”. Ez a felhasználónak gyakorlatilag olvashatatlan, ezért szükséges ezeket a BSON dokumentumokat átalakítani a felhasználó számára JSON formátumba.

3.2.1. ObjectId

Minden dokumentum létrehozása során a kiszolgáló generál egy ObjectId típusú kulcs/érték párt, aminek a kulcsa az `_id` nevet viseli, értéke pedig egy ObjectId típusú adatmező. Ez a mező összesen 12 bájtos, azaz 24 hexadecimális karakterből áll. [11] Az első 4 bájt a dokumentum létrehozásának idejét tárolja Unix-idő szerint. A következő 3 bájt az eszköz azonosítóját tartalmazza, majd 2 bájt a folyamatot azonosítja és az utolsó 3 bájt egy számláló, ami először véletlen szám generátorral jön létre, majd ha egyszerre több dokumentumot hozunk létre egyel növekszik az értéke a következőnek.

Az ObjectId elsődleges kulcsként is szolgál, és alapértelmezetten indexelt. Ha a kulcs mező alapján szeretnénk rendezni a dokumentumainkat, akkor létrehozási idő alapján rendezni sorrendbe az adatokat. A 3 bájtos számláló azért felel, hogy minden dokumentumban egyedi értéket kapjunk. További előnye, hogy használhatunk beépített függvényeket is az azonosító értelmezéséhez. Ilyen a `getTimestamp()`, ami kiírja számunkra értelmezhető formában a létrehozás dátumát. Használhatunk még `str()` metódust, ami string formává alakítja az azonosítót.

3.2.2. Indexelés

Az indexelés a MongoDB-ben hasonlóképp működik, mint a MySQL-ben. Az indexelés segítségével hatékonyabbá tehetjük a read utasításokat. Ha egy bizonyos mezőt indexelünk, később jóval rövidebb végrehajtási idő alatt juthatunk hozzá a szükséges eredményeinkhez. [11] Az indexelt mezők egy B-fa algoritmust használnak fel a gyors eredmény eléréséhez. Fő hátránya, hogy a módszer használata miatt az adatbázis mérete nő. Sok indexelt oszlop esetén a beszúrás jóval lassabb lesz, ezért erősen ajánlott, hogy csak azokat az adatokat indexeljük, amelyeket biztosan sokszor fogunk használni. Beágyazott dokumentumok esetén is használható.

A MongoDB ismeri az egyedi kulcsok használatát is. Ahhoz hogy biztosak legyünk abban, hogy nincs két ugyanolyan értékű dokumentumunk, érdemes használni a `dropDups()` nevű metódust, ami megtartja az első talált adatot, majd ha talál még egy ugyanolyan kulcsban szereplő értéket, akkor ki fogja törölni a következő dokumentumot mindenféle figyelmeztetés nélkül. Ezért fontos, hogy nagy körültekintéssel használjuk ezt a funkciót.

Az `explain()` metódushívásával fontos információkat kaphatunk egy művelet végrehajtásáról. A 2. ábra az `explain()` metódust mutatja be [11].

```

> db.phpt.find({"Hello":"world"}).explain()
{
  "cursor" : "BtreeCursor Hello_1",
  "isMultiKey" : false,
  "n" : 1,
  "nscannedObjects" : 1,
  "nscanned" : 1,
  "nscannedObjectsAllPlans" : 1,
  "nscannedAllPlans" : 1,
  "scanAndOrder" : false,
  "indexOnly" : false,
  "nYields" : 0,
  "nChunkSkips" : 0,
  "millis" : 0,
  "indexBounds" : {
    "Hello" : [
      [
        "world",
        "world"
      ]
    ]
  }
}

```

2. ábra: Explain() metódus bemutatása

A „cursor” kezdetű sor megmutatja, hogy a művelet végrehajtása során indexelt adatokból származik –e az adat. Ahhoz, hogy megtudjuk a lekérdezés során visszaadott dokumentumok számát, az n sor értéke alapján kaphatunk választ. Az nscanned sor mutatja az összesen megvizsgált dokumentumok számát, amely minél kevesebb, annál jobb keresés algoritmust használt fel a program. A millis tulajdonság a lekérdezés teljes idejét mutatja meg milliszekundumban.

3.2.3. CRUD a MongoDB-ben

A következő táblázat a négy alapvető adatbázis művelet kulcsszavát tartalmazza MongoDB és MySQL nyelven.

	MongoDB	MySQL
Beszúrás	insert	insert/create
Olvasás	find	select
Módosítás	update	update
Törlés	remove	delete/drop

A következőkben néhány alapvető művelet kódját szeretném bemutatni a MongoDB Shelljét használva. Egy utasítás kezdetét a > jel fogja reprezentálni. MongoDB-ben egy utasítás három részből áll, az egyes részeket a „.” karakter választja el. Az első rész a db karaktersorozat. Ez arra utal, hogy az aktuálisan használatos adatbázison hajtjuk végre a műveletet. Az aktuális adatbázisunkat, a >db parancs használatával tudhatjuk meg, a >show dbs mutatja az összes adatbázisunkat a serveren, míg a >use <adatbázisnév>

segítségével válthatunk az adatbázisok között. Ilyenkor megvizsgálja, hogy létezik –e az adatbázis, ha nem akkor létrehozza a felhasználó számára transzparens módon.

Az utasítás második paramétere minden esetben a collection neve. A **>show collections** művelet segítségével megtudhatjuk milyen collectionok találhatók adatbázisunkban. A következő rész pedig maga a függvény, benne az esetleges paraméterek.

3.2.3.1. Create

Ha egy dokumentumot szeretnénk beilleszteni, akkor a **>db.collectionnev.insert()** parancsot használhatjuk, ahol az insert paraméterében adhatjuk meg a dokumentum attribútumait, vagy akár egy JSON dokumentumot tároló hivatkozást. Például ha **>db.collectionnev.find({"Hello": "World", "Szulev":1992})** dokumentumot hozzuk létre. Ekkor beillesztésre kerül a dokumentum, illetve elsődleges kulcsként az automatikusan generált ObjectID.

Egy collection létrehozása hasonlóan transzparens módon történik, mint egy új adatbázis beillesztése. Amikor a parancsban egy olyan collection nevet adunk meg, ami nem létezik a MongoDB mindenféle figyelmeztető üzenet nélkül létrehozza az új collectiont.

3.2.3.2. Read

Az általános olvasási művelet a **>db.collectionnev.find()**. Ezzel szemben **findOne()** függvény esetén a legutoljára beillesztett dokumentumot fogja eredményként megadni, üres collection esetén nincs visszatérési értéke. Lehetőségünk van kereséseink szűkítésére is. A következő művelet a Bela nevű felhasználók sorait fogja eredményként visszaadni: **>db.collectionnev.find({"Nev": "Bela"}, {"Nev":true, "_id":false})**. A példában az első paraméter a keresési feltétel. Ha a második paramétert elhagyjuk, akkor a dokumentum minden kulcs/érték elemét kiírja, itt viszont csak a nevet fogja kiírni, az azonosítót nem. Mivel strukturálatlan adatokról beszélünk, előfordulhat, hogy néhány dokumentumban más értékeke is helyet kapnak. Ilyenkor egyik sem kerül kiíratásra. További keresési feltételek közé sorolhatjuk, a **\$gt** (nagyobb, mint), a **\$lt** (kisebb, mint) és a **\$gte**, **\$lte** (e = equal) paramétereket. Például a **>db.collectionnev.find({"Szulido":{"\$gt:1991 , \$lte:1993}})** kilistázza az 1992-ben született személyek adatait. Ezt a feltételt használhatjuk stringek esetén is. A **>db.collectionnev.find({"Nev": {"\$lt : "D" , \$gt : "B"}})** a C kezdőbetűjű neveket listázhatjuk ki.

Az adatmodellünk változó szerkezete miatt, szükségünk lehet olyan lekérdezésekre, ahol csak azokat a dokumentumokat szeretnénk megjeleníteni, amelyekben létezik egy bizonyos mező. Erre is lehetőségünk van a **>db.collectionnev.find({„Nev” : { \$exists : true}})** paraméter használatával.

Különösen stringek esetén hasznos lehet a **\$regex** paraméter alapján a keresésünk szűkítése. A **\$regex** segítségével kilistázhathatjuk azon dokumentumokat, amelyek egy adott értékben megtalálhatóak. A keresési feltétel lehet egy karakter vagy akár egy string is. Például a **>db.collectionnev.find({„Nev” : { \$regex : „a\$”}})** azt mutatja meg, hogy az utolsó karakter „a” legyen. Az előbb ismertetett paramétereket együttes használatra alkalmazhatjuk az **and**, illetve **or** utasítások segítségével.

3.2.3.3. Update

Az **update** utasítás alapos ismerete azért fontos, mert egy rossz paraméterezés esetén akár a teljes adatbázis tartalmát is felülírhatjuk. Ilyen típusú paraméterezést láthatunk a **>db.collectionnev.update({„Nev”: „Kovacs”}, {„Nev”: „Kiss”, „Szulev” : 1991})** példában. Ebben az esetben minden Kovacs nevű felhasználót Kiss nevére cserél, és a születési évet is minden dokumentumban megváltoztatja, ha létezik ilyen mezőnév. Ellenkező esetben létre fogja hozni. Fontos tudni, hogy ilyenkor az **ObjectId**-n és a **Nev**, **Szulev** mezőkön kívül az összes kulcs/érték párt eldobja a dokumentumon belül.

A **\$set** parancs segítségével hozzáfűzhetünk a dokumentumhoz bizonyos kulcs/érték párokat. A **>db.collectionnev.update({„Nev”: „Kiss”}, {\$set: {„Kor” : 30}})** az eddigi mezőkhöz hozzáfűzi az életkor mezőjét. Növelhetjük az **\$inc** parancs segítségével egy mező értékét a **>db.collectionnev.update({„Nev”: „Kiss”}, {\$inc : {„Kor”:1}})** minta alapján. Hasonlóképpen a **\$dec** használatával csökkenthetjük egy mező értékét. Ha több dokumentumot szeretnénk egyszerre módosítani, akkor a **\$multi:true** paranccsal tehetjük meg.

A két megoldás hasonlósága, hogy az **ObjectId** egyik esetben sem változik, illetve változtatható meg.

3.2.3.4. Delete

Egy adott collection teljes tartalmát a **>db.collectionnev.remove({})** utasítással törölhetjük ki. [12 old.: 23-44] Ilyenkor az indexek is eltávolításra kerülnek. Lehetőségünk van olyan

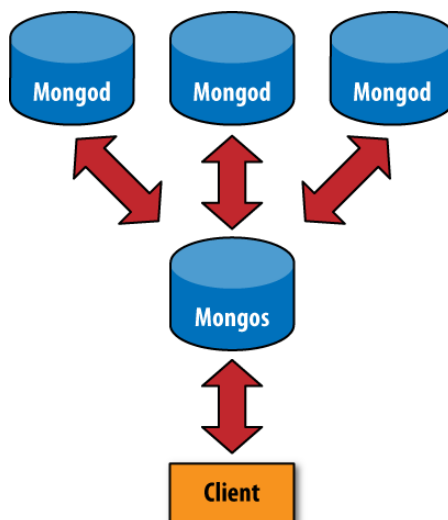
dokumentumokat eltávolítani az adatbázisból, amelyek egy adott feltételnek tesznek eleget. A feltételeknek ugyanazon értékeket megadhatjuk, mint az olvasási műveletek esetében. A következő példában az M kezdőbetű utáni szavakat távolítja el a collectionból: `>db.collectionnev.remove({"Nev": {"$gt": „M”}})`. A **drop** utasítás használatával az indexek megmaradnak.

3.3. Skálázhatóság a MongoDB-ben

A MongoDB fejlesztői a horizontális skálázhatóságot helyezték előtérbe megvalósítás során a vertikális megoldás helyett. Ilyenkor a teljesítmény növelése érdekében új eszközök is kiszolgálhatják az adatbázis műveleteket, amely eszközök akár fizikailag is különböző helyen lehetnek. Ezt az eljárást shardingnek nevezik.

Akkor érdemes használni a shardingot, ha valamilyen fizikai korlátba ütközünk, például megtelt a merevlemez, vagy éppen kevés az erőforrás az adatbázis hatékony működéséhez. A sharding előnye, hogy több kiszolgáló esetén, több adat kerül a memóriába, ami csökkenti a lekérdezési műveletek végrehajtási idejét. A shardingnek két fajta implementációja létezik, vannak az úgynevezett manuális, illetve automatikus sharding eljárások. Manuális shardingot az adatbázis kezelők zöme támogatja elnevezésbeli különbségekkel. Ennek az eljárásnak a konfigurációja nehéz, hosszú ideig tart, illetve nem tartalmaz hibakezelést.

A MongoDB az automata shardingot használja. Ez a módszer lényegesen leegyszerűsíti a terheléeloszlás használatát. Az architektúráis megvalósítását a 3. ábra mutatja:



3. ábra: Sharding fizikai felépítése

Az eljárás lényege, hogy a collectiont kisebb részekre, úgynevezett chunkokra bontjuk. A chunkokat pedig tárolhatjuk különböző kiszolgálókon, amiket a fenti ábrán a Mongod jelöl. Nem szükséges tudni a kiszolgálónak, hogy mely chunkot tartalmazza, hanem a szerverek felett egy Router kerül bevezetésre, ami a Mongos. Ehhez az útválasztóhoz érkeznek be a kliensektől érkezett lekérdezések. Ha olvasási műveletet hajtunk végre, a Router tudni fogja hol található a keresett adat, beszúrás előtt pedig egy úgynevezett autobalancert futtat, amely megvizsgálja, melyik kiszolgálón optimális az adattárolás.

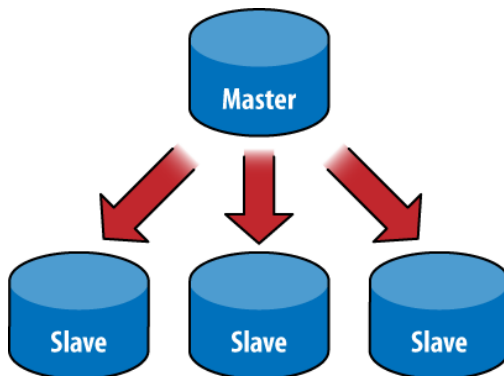
A Mongos a balancer műveletet a shardkey nevű paraméter alapján hajtja végre. Ebben a paraméterben egy mezőnevet kell megadnunk, ami alapján a balancer a collectiont több fizikailag különböző helyre bontja szét. Például ha osztályzatok alapján szeretnénk egy collectiont öt Mongod kiszolgálón tárolni, akkor a különböző jegyeket különböző szerveren tárolja. A balancer azonban az adatforgalmat is vizsgálja, azaz a gyakran használt adatokat kisebb chunkban tárolja. Ez akkor fontos, amikor egy kiszolgálóra több mint egy collection jut. Ilyen esetben gyakrabban lekérdezett adatokat külön chunkba teszi a Mongos.

Sharding eljárás bevezetése általában akkor lesz szükséges, amikor már az adatbázisunk mérete jelentősnek mondható. [13] Ennek a problémának a figyelembe vételével alakították ki úgy a módszert a fejlesztők, hogy akár egy meglévő adatbázist is chunkokra bonthassuk. A collectiont timestamp alapján is részekre bonthatjuk. Ebben az esetben egy megadott létrehozási időnél korábbi és későbbi dokumentumok kerülnek külön egy-egy chunkba. Ennek az a hátránya, hogy az újonnan létrehozott dokumentumot egy collectionbe tárolja, ezzel felborul a balance a szerverek között, ha huzamosabb ideig egy helyen tároljuk az új dokumentumokat. Ennek a megoldása az, hogy meghatározott időközönként más és más chunkban fogja tárolni az újabb bejegyzéseket. Egy esetleges lekérdezés esetén a Mongos router tárolja, hogy melyik időintervallumban melyik Mongod-ben vannak a dokumentumok beillesztve.

3.4. Replikáció a MongoDB-ben

Az adatbázis replikációja egy nagyon lényeges kérdés a tervezés során, mert ezen eljárás segítségével javíthatjuk adatbázisunk rendelkezésre állását, illetve biztonsági mentéseket készíthetünk adatainkról.

A replikációnak két fajtáját ismertetem. Az architektúrális felépítés mindkét esetben megegyezik, a 4.ábrán látható módon.



4. ábra: Replikáció fizikai felépítése

A master-slave replikáció fő feladata, hogy a master adatbázisról valós időben készüljön egy vagy több biztonsági mentés, amelyeket egy esetleges meghibásodás esetén elérhessen (hálózati hiba, hardver hiba). Ezt a megoldást a mai adatbázis kezelők nagy része tartalmazza a tartósság és rendelkezésre állás javítása érdekében, azonban egy esetleges meghibásodás esetén a fejlesztőnek sok konfigurációs utasítást kell végrehajtani ahhoz, hogy egy slaveből master legyen. További beállításokat kell alkalmazni, ha a master később a hiba megoldása után újra elsődleges adatbázisként működjön a hálózatban.

A replicaset a MongoDB megoldása a replikáció minőségének javítására. A replicaset alapja az előbb ismertetett master-slave megoldás. Működését, ha a shardinghoz szeretném hasonlítani, akkor ezt a megoldását auto master-slave replikációnak nevezném. Ez azt jelenti, hogy a fejlesztőnek a lehető legkevesebb adminisztrációs tevékenységet kell végrehajtania a sikeres replikációhoz. A gyakorlatban ez annyit jelent, hogy kevés konfigurálási beállítás szükséges a replikáció működéséhez[12]. A legfontosabb különbség az előző eljárásához képest az, ami a master adatbázis rendelkezésre állásának hiánya esetén történik. Ilyen esetben az adminisztrátor beavatkozása nélkül automatikusan átveszi valamely slave vagy másodlagos adatbázis az elsődleges szerepét. Abban az esetben, ha több másodlagos adattárolónk van, akkor egy választás alapján dől el, melyik legyen az elsődleges adatbázis. A másodlagos adatbázisoknak vannak prioritási szintjeik. A legmagasabb prioritású lesz az elsődleges adatbázis, ha azonban több ilyen adatbázis van, akkor a legutoljára szinkronizált másodlagos adatbázis kapja ezt a szerepet. Abban az esetben, amikor újra rendelkezésre áll az eredeti elsődleges adatbázis, akkor először

szinkronizál a jelenlegi elsődleges adatbázissal, majd ha a szinkronizálás sikeres volt, akkor újra átveszi az elsődleges adattároló szerepét.

3.5. MongoDB tárolási módja, technikai korlátai

Egy dokumentum maximális mérete összesen 16MB lehet. Ezt a korlátot multimédia adatok tárolása esetén könnyen túlléphetjük. A GridFS eljárás segítségével azonban a felső korlátot meghaladó dokumentumokat kisebb darabokra bonthatjuk. Ilyenkor két collection-ben tárolódnak az adatok. Egyikben a több részre osztott dokumentum, míg a másikban metaadatok találhatók.

A MongoDB alapértelmezetten a data könyvtárban, ezen belül az adatbázis nevével megegyező alkönyvtárban tárolja a szükséges fájlokat. A dokumentumokat az adatbázison belül az aktuális collection nevű fájlban tárolja. Az adatbázis méretétől függ, hogy mennyi állományban tárolja a MongoDB az adatokat. [14] Először a <collectionnev>.0 fájlban fogja tárolni az adatokat. Ha eléri az adatbázis mérete a 64 MB-ot, akkor új fájl jön létre, amelynek 1 lesz a kiterjesztése, és mérete a duplájára nő. Ez egészen a 2 GB-os méretkorlátig nő. Ilyen esetben a többi fájl is 2 GB-ot marad. Ezen tárolási mód működési elve, hogy előre lefoglalja a fent említett lemezterületeket, amikor egy új fájlt hozunk létre. Ez azt jelenti, hogy nem minden esetben fogja a teljes lefoglalt területet adattárolásra használni. Az adatbázisban megtalálhatóak az úgynevezett journal fájlok. Ezen állományok felelnek a naplózás működéséről, lehetőséget biztosítva egy hirtelen leállás esetén az adatbázis helyreállítására, illetve a leálláskor sikertelenül beszúrt dokumentumok hozzáadására az adatbázisunkhoz.

5. Összehasonlító mérési eredmények

5.1. Felhasznált eszközök

Méréseimet a következő laptopon hajtottam végre:

- Processzor: AMD Athlon X2 P320 2,1 GHz
- Memória: 3GB 1066 DDR3
- Merevlemez: 320GB SATA 5400 rpm
- Videovezérlő: ATI Radeon HD4250
- Telepített operációs rendszer: Windows 8.1

Munkám során helyi szervert telepítettem, amelynek köszönhetően az eredményeket nem befolyásolja a hálózat aktuális sebessége, így a hálózattól független eredményt kapunk.

Ehhez a 2.4-es verziójú WampServert telepítettem fel.

Ez az alkalmazás tartalmazza az Apache 2.4.4-es, a MySQL 5.3-os és a PHP 5.4-es verzióját.

A PHP-hez hozzáadtam egy php_mongo 1.4.4.dll nevű kiterjesztést, ami egy interfészként működik a PHP és a Mongo 2.4.9-es verziószámú adatbázisszerver között.

5.2. Adatbázisaim

Adatbázisom rekordjai illetve dokumantumai egy azonosító számból, ami MongoDB esetén az ObjectId, egy 200 karakter hosszú véletlenszerű betűkből álló szó, és egy 100 milliárdos nagyságrendű véletlenszám-generátorral előállított negatív szám.

Mindkét adatbázisban körülbelül 1 700 000 bejegyzést hoztam létre, amely megfelelő nagyságrendű, hogy összehasonlíthassuk az esetleges sebesség különbséget a lekérdezések során.

Az 5. ábra egy dokumentumot jelenít meg a MongoDB adatbázisból.

```
> db.php.find().pretty()
{
  "_id" : ObjectId("5328860d8396608c0b042fd1"),
  "randKar" : "qlgtDGnoIaal9UQaSv7bo3egE7TK8PjsNLdK710t3bPY51u5b7va0z62VkJZ31so42y2ptJ7WgznYU91om5Mv6tRb4TYKMrn39IJrH0jLlkNmBkq4prVegKtH6QQ0p0dq9gjRNhnYjs67aTI71bVkipYAZazfUzV90bLDL7jy9YgeCPRx9hPNdqmRLRb4iGwgKt5Nck7v",
  "randSzam" : -1731150382
}
```

5. ábra: Egy dokumentum a MongoDB adatbázisból

A 6. ábra egy rekordot ábrázol a MySQL adatbázisból.

id	ranKar	randSzam
1	daUPmZPB6laaXr8IRMttiNnpe6Av8NizXtfYccDzOBdFhKa9xf...	-996987619

6. ábra: Egy rekord a MySQL adatbázisból

Többféle mérést is végeztem, amellyel össze tudom hasonlítani a MongoDB és a MySQL adatbázis feldolgozási sebességét.

Először az adatbázis műveletek sebességét egy erre a célra készített PHP oldalon végzett lekérdezésekkel mértem le.

5.3. A PHP Webalkalmazás ismertetése

A két adatbázisnak két különböző Weboldalt hoztam létre, amelyek egymással megegyeznek, kivéve az adatbázis kapcsolatokat.

A 9. ábrán látható az *index.php* részlete. Ezen keresztül tudjuk a különböző lekérdezéseinket elindítani. Az oldal legfelső részén illeszthetünk be annyi sort az adatbázisba, amennyit megadunk. Ha rákattintunk az **insert random** gombra, a beszúrás megkezdődik, és kiírja a végrehajtáshoz szükséges időt.

A következő funkció egy **delete** gomb, ami az adatbázis teljes tartalmának törlésére szolgál. Ha sikerült a törlő művelet, akkor szintén kiírja a felhasznált időt.

Ezután található az azonosító alapján történő keresés. Az input mezőben meg kell adnunk MySQL esetén a rekord azonosítószámát Mongóban pedig az ObjectID-t. Ha talált a kritériumnak megfelelő elemet, akkor kiírja az összes adatát, majd a művelethez felhasznált időmennyiséget is.

Az utolsó vizsgált lekérdezés az oldalon, amikor egy random generált szám alapján keresünk. Ha megadtuk a mezőben a számot, akkor a gomb megnyomása után elindul a keresés, és kiírja a kritériumoknak megfelelő elemeket, és a futási időt is.

Ha megadom, hogy a 4210. elemet jelenítse meg, a 7. ábrán a MySQL adatbázis lekérdezés eredménye tekinthető meg.

```
Id: 4210
RanKar:
xZBivxovFsQTke4EmODx6FDJpLGZBARvx6LZyLtHuyCjIVf42RZFPPUmVNDQFeelT5jzXRWKHvKBsuOl5NIGVRzROGXA6JY3DXd1KQMghgMAWKvwAPAOmAe5
RandSzam: -556526866
A MySQL 0.034 másodpercig tartott!
```

7. ábra: Egy lekérdezés eredménye MySQL adatbázisból

A 8. ábrán egy ObjectID alapján történt lekérdezés eredményét listázta ki a MongoDB adatbázisból.

```
array (size=3)
  '_id' =>
    object (MongoId) [8]
      public '$id' => string '5302665d839660140c007068' (length=24)
  'randKar' => string 'ABGZvQlnkM90SgDwc1e67MbWC24TGoOjMXClRieoqwQFpdwJuvvi4UuGsb1WB8PpOB7VLCXFPVgz2h8E1hkrSUGEOBwvmpEISJvDubxjzEjoea61q2S8xgQCjHjXQxa5Itiv2Kfbo3QzSDAD5bpR7'
  'randSzam' => int -696027632
```

A művelet 0.001 másodpercig tartott!

8. ábra: Egy lekérdezés eredménye MongoDB adatbázisból

Az index. php egy képernyőképe, ami Mongo esetében is megegyezik:

Adatbázis használata MySql segítségével

A következő gomb lenyomásával annyi dokumentumot illeszthetünk be, amennyit megadunk a mezőben.

Dokumentumok száma:

Adatbázis teljes tartalmának törlése

Adja meg hányadik elem adatait jelenéítem meg:

Dokumentumok száma:

9. ábra: Index.php részlete

A 10. ábrán a MySQL adatbázisba történő beszúrási művelet forráskódja látható.

```
12 $characters =
    '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
13 $sex = 200;
14 $numOfRec = $_GET["szam"];
15 for ($x=0; $x<=$numOfRec; $x++)
16 {
17     $randomString = '';
18     for ($i = 0; $i < $sex; $i++) {
19         $randomString .= $characters[rand(0, strlen($characters) - 1)];
20     }
21     $rndNum = Rand(0,543445343456);
22     $starttime = microtime(true);
23     mysql_query("insert into targyak(ranKar,randSzam)
    values ('$randomString','$rndNum')");
24     $endtime = microtime(true);
25     $duration = $endtime - $starttime;
26     $qtime = $qtime + $duration ;
```

10. ábra: MySQL beszúrási művelete

Ugyanezen művelet forráskódja MongoDB adatbázis használata esetén a 11. ábrán látható.

```
22  $characters =  
    '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';  
23  $ex = 200;  
24  for ($x=0; $x<=$numOfDoc; $x++)  
25  {  
26      $randomString = '';  
27  for ($i = 0; $i < $ex; $i++) {  
28      $randomString .= $characters[rand(0, strlen($characters) - 1)];  
29  }  
30  
31      $a = array ('randKar' => $randomString, 'randSzam' =>  
    Rand(0,543445343456));  
32      $starttime = microtime(true);  
33      $collection->insert($a);  
34      $endtime = microtime(true);  
35      $duration = $endtime - $starttime;  
36      $qtime = $qtime + $duration;
```

11. ábra: MongoDB beszúrás művelete

Az első két sorban a string generálásához felhasznált betűket és számokat definiálom, majd megadom a szó hosszát. Ezután felhasználok a bekért értéket, ami megadja az új rekordoknak a számát, azaz ennyiszer fog lefutni a ciklusmag. Először generál egy véletlen karaktersorozatot, majd egy számot. Utána végrehajtja a lekérdezést, aminek az idejét a microtime utasítással kapom meg, amit tárolok a qtime nevű változóban, és a ciklusból kilépve kiírok a képernyőre.

5.4. Mérési műveletek

Első mérésem során a rekordok, illetve dokumentumok beszúrási sebességét mértem le, különböző mennyiségű adatok beszúráásával. A következő két mérésem az olvasási művelethez kapcsolódó végrehajtási időket vizsgálta. Első lekérdezésnél a MongoDB esetében az ObjectId alapján kerestem. Ilyen esetben, mivel az ObjectId egy elsődleges kulcs tulajdonságával is rendelkezik, ezért vagy 1, vagy 0 dokumentumot fog eredményként megadni. A MySQL lekérdezésben egy megadott ID alapján történik a szűrés.

A táblázatokban erre a lekérdezésre keresés adott kulcs alapján néven fogok hivatkozni.

A másik lekérdezésben a generált véletlen szám alapján történik a keresés.

Ebben az esetben nem tudjuk előre mennyi eredményt fogunk kapni, mert több rekordban/dokumentumban is lehet ugyanolyan értékű szám, ezért mindkét esetben a teljes adatbázis tartalmát meg kell vizsgálnunk.

A táblázatokban erre a lekérdezésre keresés számérték alapján néven fogok hivatkozni.

A két lekérdezés fő különbsége a visszaadott eredmények mennyisége. Az első keresés esetén maximum 1 eredménye lehet, míg a második esetben nincs ilyen mennyiségi korlát.

Az1. táblázatban a beszűrési, illetve az előbb bemutatott két keresési művelet eredményeit ismertetem, táblázat formájában. Ezen eredmények 1 700 000 bejegyzést tartalmazó adatbázis feldolgozásakor keletkeztek.

5.5. A mérések eredményei

<u>adatbázis</u>	<u>művelet</u>	<u>beszűrt sorok száma</u>	<u>eredményként visszaadott sorok száma</u>	<u>művelethez végrehajtott idő (s)</u>	<u>egy soron végrehajtott művelet ideje (s)</u>
Mongo	insert	1		0,002	0,002
MySQL	insert	1		0.224	0,224
Mongo	insert	1 000		0.867	0,000867
MySQL	insert	1 000		40.958	0,040958
Mongo	insert	10 000		8.560	0,000856
MySQL	insert	10 000		379.87	0,037987
Mongo	insert	25 000		20.890	0,0008356
MySQL	insert	25 000		1060.411	0,04241644
Mongo	insert	50 000		54.132	0,00108264
MySQL	insert	50 000		1989.847	0,039797
Mongo	insert	75 000		65.475	0,000873
MySQL	insert	75 000		2962.71	0,0395028
Mongo	insert	100 000		82.912	0,00082912
MySQL	insert	100 000		4075.16	0,0407516
<u>Keresés adott kulcs alapján</u>					
Mongo	find		1	0.52	
MySQL	read		1	0.103	
Mongo	find		0	0.00	
MySQL	read		0	0.01	

<u>Keresés számérték alapján</u>					
Mongo	find		változó	2.41	
MySQL	read		változó	9.84	

1. táblázat: Mérések eredményei 1 700 00 bejegyzés méretű adatbázisban

Amikor az adatbázisaimban körülbelül 1000 rekord kapott helyet, akkor az eredményeim a 2. táblázatban láthatóak:

<u>adatbázis</u>	<u>művelet</u>	<u>beszúrt sorok száma</u>	<u>eredményként visszaadott sorok száma</u>	<u>művelethez végrehajtott idő(s)</u>	<u>egy soron végrehajtott művelet ideje(s)</u>
Mongo	insert	1		0.02	0.02
MySQL	insert	1		0.231	0.231
Mongo	insert	1 000		0.925	0.000925
MySQL	insert	1 000		41.245	0.041245
Mongo	insert	10 000		9.48	0.000948
MySQL	insert	10 000		387.84	0.038784
<u>Keresés adott kulcs alapján</u>					
Mongo	find		1	0.001	
MySQL	read		1	0.103	
Mongo	find		0	0.00	
MySQL	read		0	0.01	
<u>Keresés számérték alapján</u>					
Mongo	find		változó	0.021	
MySQL	read		változó	0.002892	

2. táblázat: Mérések eredményei 1000 bejegyzés méretű adatbázisban

A beszúrási művelet eredményei a 12. ábrán is megtekinthetők.



12. ábra: Beszúrási műveletek összehasonlítása

5.6. Mérés Apache Benchmark felhasználásával

Az Apache Benchmark segítségével a HTTP Webszerver teljesítményét mérhetjük. Az eszközt tartalmazza az Apache binary. Eredményként válaszidőt fog visszaadni, ami minél kevesebb, annál gyorsabban szolgálta ki a szerver a klienst. Maga a benchmark program GET kéréseket küld a szerver felé. A kérések számát paraméterként adhatjuk meg. A `-n` tulajdonság segítségével adhatjuk meg a lekérdezések számát, míg a `-c` jelöli a konkurens, azaz egy időben történő kérések számát. Ezt a két paramétert vizsgálva készítettem el a két adatbázis tesztjét.

A 13. ábra egy Apache Benchmark parancsot mutat, ahol 50 lekérdezés idejét méri.

```
D:\wamp\bin\apache\Apache2.4.4\bin>ab -n 50 localhost/myMysqlSite/firstQuery.php
```

13. ábra: Apache Benchmark működése

A 3. táblázat az Apache Benchmark által mért három művelet eredményeit mutatják, amelyek tartalmaznak konkurens lekérdezéseket is.

<u>Beszúrási lekérdezés eredményei</u>						
<u>adatbázis</u>	<u>művelet</u>	<u>kérések száma</u>	<u>ebből konkurens kérés</u>	<u>összes művelethez végrehajtott idő (s)</u>	<u>egy lekérdezés minimális ideje (ms)</u>	<u>egy lekérdezés maximális ideje (ms)</u>
Mongo	insert	10	0	0.155	12	23
MySQL	insert	10	0	0.737	58	126
Mongo	insert	25	0	0.338	11	15
MySQL	insert	25	0	1.923	40	144
Mongo	insert	50	0	0.663	11	16
MySQL	insert	50	0	3.185	42	146
Mongo	insert	100	0	1.33	11	22
MySQL	insert	100	0	6.249	43	217
Mongo	insert	200	0	2.658	11	18
MySQL	insert	200	0	13.99	40	357
Mongo	insert	10	5	0.095	17	66
MySQL	insert	10	5	0.239	69	162
Mongo	insert	25	5	0.215	16	114
MySQL	insert	25	5	0.479	59	146
Mongo	insert	50	10	0.425	45	139
MySQL	insert	50	10	0.749	59	251
Mongo	insert	100	10	0.855	26	150
MySQL	insert	100	10	1.425	61	145
Mongo	insert	200	20	1.74	45	104
MySQL	insert	200	20	2.28	108	328

<u>Keresés adott kulcs alapján</u>						
<u>adatbázis</u>	<u>művelet</u>	<u>kérések száma</u>	<u>ebből konkurens kérés</u>	<u>művelethez végrehajtott idő (s)</u>	<u>egy lekérdezés minimális ideje (ms)</u>	<u>egy lekérdezés maximális ideje (ms)</u>
Mongo	find	10	0	0.134	14(ms)	18
MySQL	select	10	0	0.164	10	49
Mongo	find	25	0	0.319	12	18
MySQL	select	25	0	0.395	9	42
Mongo	find	50	0	0.633	12	16
MySQL	select	50	0	0.781	9	35
Mongo	find	100	0	1.254	11	17
MySQL	select	100	0	1.565	8	34
Mongo	find	200	0	2.516	11	17
MySQL	select	200	0	3.142	8	26
Mongo	find	10	5	0.078	15	70
MySQL	select	10	5	0.098	18	50
Mongo	find	25	5	0.189	16	89
MySQL	select	25	5	0.235	16	70
Mongo	find	50	10	0.379	30	113
MySQL	select	50	10	0.466	40	124
Mongo	find	100	10	0.790	32	237
MySQL	select	100	10	0.972	37	120
Mongo	find	200	20	1.622	55	281
MySQL	select	200	20	1.981	80	299

<u>Keresés számérték alapján</u>						
<u>adatbázis</u>	<u>művelet</u>	<u>kérések száma</u>	<u>ebből konkurens kérés</u>	<u>művelethez végrehajtott idő</u>	<u>egy lekérdezés minimális ideje</u>	<u>egy lekérdezés maximális ideje</u>
Mongo	find	10	0	24.586	2433	2489
MySQL	find	10	0	126.204	9001	22097
Mongo	find	25	0	62.420	2377	3195
MySQL	find	25	0	261.994	8942	13518
Mongo	find	50	0	120.213	2346	2523
MySQL	find	50	0	526.144	9203	18356
Mongo	find	100	0	241.069	2342	2472
MySQL	find	100	0	1054.227	8910	17206
Mongo	find	200	0	480.763	2335	2805
MySQL	find	200	0	2008.140	8956	20542
Mongo	find	10	5	15.195	7160	7952
MySQL	find	10	5	35.126	15824	18588
Mongo	find	25	5	35.980	7138	7548
MySQL	find	25	5	86.8	15525	16989
Mongo	find	50	10	65.916	11766	15375
MySQL	find	50	10	147.874	n/a	n/a
Mongo	find	100	10	125.532	10488	13702
MySQL	find	100	10	292.755	n/a	n/a
Mongo	find	200	20	247.651	16551	29271
MySQL	find	200	20	587.418	n/a	n/a

3. táblázat: Apache Benchmark által mért eredmények

6. Konklúzió

A mérési eredmények alapján megállapítható, hogy az olvasási műveletek ideje függ az adatbázis méretétől. Abban az esetben, ha a lekérdezések között konkurens kérések is szerepelnek, akkor a MongoDB hatékonysága csökkenni kezd. Ennek oka, hogy a MongoDB sorosan szolgálja ki a beérkező kéréseket, legalábbis abban az esetben, ha egy kiszolgálónk van. Ilyenkor a vertikális skálázhatóságot használó MySQL bizonyul jobb választásnak. Ha az összes lekérdezés és a konkurens lekérdezések aránya körülbelül 70%, akkor a MySQL hatékonyabban szolgálja ki a beérkező kéréseket.

Az eredmények ismeretében megállapítható az is, hogy MongoDB adatbázist különösen akkor érdemes használni, ha nagy mennyiségű (millió nagyságrendű) adatot tárolunk az adatbázisban. Az ilyen adatbázison végrehajtott olvasási műveletek teljesítménye nagyságrendekkel jobb lesz.

Sztruktúrálatlan adatok halmazát relációs adatbázisban tárolni nem optimális, hiszen az adatbázis minden sorához hozzárendeli az oszlopokban meghatározott mezőket, akkor is, ha az adott mező üresen marad. Ha az adatbázis szerkezetét tervezés után módosítani kell, a MySQL esetén nagyobb költséget jelent.

MongoDB segítségével csökkenthetjük a relációs adatbázis tábláinak számát beágyazott dokumentumok használatával. Fontos azonban megjegyezni, hogy a beágyazottság mélysége egyenesen arányos a redundancia nagyságával. Túlzott redundancia esetén az adatbázis teljesítménye csökken. További előnye a MongoDB-nek, hogy összességében könnyebb tervezni, felügyelni az adatbázist, ebből kifolyólag esetlegesen kevesebb szakemberre van szükség.

Természetesen, hátrányai is vannak a MongoDB-nek. Legfőbb hátránya, hogy nem használja a tranzakciókat. Ebből kifolyólag nem használhatjuk olyan adatok tárolására, amelyek megkövetelik az adatbázis integritásának megőrzését. Ennél fogva például egy bank ügyfeleinek átutalásait továbbra is relációs adatbázisban érdemes végrehajtani.

MongoDB esetén egy adatbázis mérete többszörös a MySQL-hez képest. Kevés, jól használható grafikus segédprogramot használhatunk adatbázisunk menedzseléséhez.

Ezen szempontokat figyelembe véve érdemes döntenünk a megfelelő adatbázis kezelő használatáról.

MongoDB adatbázist naplózási bejegyzések tárolására hatékonyan alkalmazhatjuk, illetve egyre több fejlesztő fogja blogját MongoDB-n keresztül tárolni. Szenzoros mérések

eredményeinek tárolására is hatékony megoldást nyújt a MongoDB. Ezen alkalmazásoknak a relációs adatbázisnál nagyságrendekkel rövidebb végrehajtási idő miatt célszerű ilyen módon tárolni adatait. Azoknak a nagyvállalatoknak is érdemes a relációs adatbázis helyett ezt a megoldást választani, akiknek fontos a nagyfokú skálázhatóság, illetve osztott rendszeren keresztül fogják kiszolgálni a kéréseket. Az eBay média metaadatok tárolására használja a MongoDB-t.

Összetett probléma esetén nem érdemes használnunk a MongoDB rendszerét. Sok adatbázistábla felhasználása esetén a beágyazott dokumentumok miatt nagyfokú redundancia léphet fel. Érdemes lesz azonban a hibrid rendszerek alkalmazását megvizsgálni, ugyanis sok összetett adatbázis egy részét érdemes lehet relációs adatbázisban, más részét pedig NoSQL adatbázisban tárolni, így optimális sebesség érhető el.

7. Összefoglalás

A Web világszintű elterjedése az adatbázisok méreteinek növekedését és nagyfokú terhelhetőségét vonja maga után. Ezen szempont figyelembe vételével érdemes olyan adatbáziskezelő-rendszereket fejleszteni, amelyek a lehető legrövidebb végrehajtási idővel hajtják végre az adott lekérdezéseket. Ahhoz, hogy tudjuk melyik rendszer lenne számunkra a leghatékonyabb, érdemes méréseket végrehajtani különböző adatbázis kezelők felhasználásával. Dolgozatom során a MySQL és a MongoDB adatbázisok teljesítményét hasonlítottam össze, egy PHP nyelven írt Webalkalmazás segítségével.

Méréseim során megállapítottam, hogy az általam vizsgált lekérdezéseket a MongoDB általánosságban rövidebb végrehajtási idő alatt hajtotta végre ugyanazon tesztkörnyezetben. A beszúrás művelet elvégzéséhez körülbelül a MySQL-nek körülbelül 35-ször több időre volt szüksége, mint a MongoDB-nek. Az egy sor beszúrására felhasznált idő akkor sem változott jelentős mértékben, amikor 10 000 vagy akár 100 000 új bejegyzést illesztettem az adatbázisba. Mindkét mérést két adatbázisban teszteltem. Az első esetben körülbelül 1 700 000 bejegyzést tartalmazott az adatbázis, míg a másik adattároló körülbelül 1000 sort tartalmazott. Megállapítottam, hogy számottevő különbséget nem tapasztaltam a különböző méretű adatbázisok használata során. Az olvasási műveletek esetén a különböző méretű adatbázisok használata során más adatbázis kezelő bizonyult hatékonyabbnak. Minél nagyobb az adatbázis mérete annál hatékonyabb megoldásnak tűnik a MongoDB. Konkurens lekérdezések vizsgálata során mindegyik művelet esetén megállapítható, hogy a MongoDB előnye csökken a MySQL-hez képest, ha nő a konkurens lekérdezések aránya az összes lekérdezéssel szemben, 70%-os arány felett a MySQL tűnik hatékonyabb megoldásnak.

A vizsgálatok után többféle konklúziót sikerült levonnom: tömören összefoglalva a MongoDB használata bizonyos problémátípusok esetén kifejezetten ajánlott, de akadnak továbbra is olyan lehetőségek, amikor a hagyományosnak tekinthető RDBMS rendszerek összességében jobb választásnak bizonyulnak.

8. Ábrajegyzék

1. ábra: MongoDB egy dokumentuma JSON formátumban	11
2. ábra: Explain() metódus bemutatása	13
3. ábra: Sharding fizikai felépítése (Forrás: [12] old.:144, 10-2.ábra)	16
4. ábra: Replikáció fizikai felépítése(Forrás: [12] old.:128, 9-2.ábra)	18
5. ábra: Egy dokumentum a MongoDB adatbázisból.....	20
6. ábra: Egy rekord a MySQL adatbázisból	21
7. ábra: Egy lekérdezés eredménye MySQL adatbázisból	21
8. ábra: Egy lekérdezés eredménye MongoDB adatbázisból	22
9. ábra: Index.php részlete.....	22
10. ábra: MySQL beszúrás művelete.....	22
11. ábra: MongoDB beszúrás művelete.....	23
12. ábra: Beszúrási műveletek összehasonlítása.....	26
13. ábra: Apache Benchmark működése	26

9. Irodalomjegyzék

1. *DB-Engines Ranking*. [Online] [Hivatkozva: 2014. április 6.] <http://db-engines.com/en/ranking>.
2. A NoSQL világa – 1. rész: Bevezető. *Weblabor*. [Online] 2010. július 19. [Hivatkozva: 2014. március 30.] <http://weblabor.hu/cikkek/nosql-bevezeto>.
3. **Strauch, Christof**. [Online] 2014. [Hivatkozva: 2014. április 10.] <http://www.christof-strauch.de/nosql dbs.pdf>.
4. A Brief History of NoSQL. *All About the Code*. [Online] 2010. március 16. [Hivatkozva: 2014. március 26.] <http://blog.knuthaugen.no/2010/03/a-brief-history-of-nosql.html>.
5. **Márton, Trencsényi**. NoSQL adatbázisok. *Adatbázisok Oktatási Labor*. [Online] 2012. szeptember 25. [Hivatkozva: 2014. április 5.] <https://db.bme.hu/~gajdos/2012adatb2/3.%20eloadas%20NoSQL%20ppt.pdf>.
6. **Misángyi, Balázs**. NoSql 1 – a CAP elmélet. *delZubu blog*. [Online] 2013. november 28. [Hivatkozva: 2014. április 2.] <http://delzubu.wordpress.com/2013/11/28/nosql-1-a-cap-elmelet/>.
7. **Seguin, Karl**. *The Little MongoDB Book*. hely nélkül. : Github.com, 2014.
8. **Rege, Gautam**. *Ruby and MongoDB Web Development Beginner's Guide*. Birmingham : Packt Publishing Ltd., 2012.
9. **Geetha Manjunath Dinkar Sitaram**. *Moving to the Cloud*. Waltham, Massachusetts : Elsevier, 2012, old.: 251-253.
10. **Javier Espinazo Pagán, Jesús García Molina**. ScienceDirect.com. *Querying large models efficiently*. [Online] 2014. január 20. [Hivatkozva: 2014. április 5.]
11. [Online] [Hivatkozva: 2014. április 2.] <http://api.mongodb.org/java/current/org/bson/types/ObjectId.html>.
12. **Jim R. Wilson Eric Redmond**. *Seven Databases in Seven Weeks*. Dallas, Texas : The Pragmatic Bookshelf, 2012, old.: 151-158.

13. **Michael Dirolf Kristina Chodorow.** *MongoDB: The Definitive Guide*. Sebastopol : O'Reilly Media, Inc., 2010.

14. Scaling MongoDB – Know your sharding Kung Fu. *comSysto blog*. [Online] 2012. december 5. [Hivatkozva: 2014. március 30.]

<http://blog.comsysto.com/2012/12/05/scaling-mongodb-know-your-sharding-kung-fu/>.

15. FAQ: MongoDB Storage. *MongoDB*. [Online] 2014. április 4. [Hivatkozva: 2014. március 25.] <http://docs.mongodb.org/manual/faq/storage/#faq-disk-size>.

16. explain.explain() — Understanding Mongo Query Behavior. *MongoHQ*. [Online] 2013. február 5. [Hivatkozva: 2014. április 6.] <http://blog.mongohq.com/explain-explain-understanding-mongo-query-behavior/>.