Budapesti Corvinus Egyetem
Gazdálkodástudományi Kar
E-Business Tanszék

DOKUMENTUM-ORIENTÁLT ADATBÁZISOK

Készítette: Oroszi Róbert
Gazdaságinformatikus szak
E-Business szakirány
2011

Konzulens:
Mohácsi László

Tartalomjegyzék

Rövidítésjegyzék

Apache Software Foundation

: Az alapítvány open-source termékek fejlesztésével, a termékekhez kapcsolódó támogatás nyújtásával foglalkozik.
Az alapítványhoz olyan termékek tartoznak, mint az Apache webszerver, a CouchDB vagy a Cassandra.

Apache webszerver

: Az egyik legismertebb open-source, keresztplatformos webszerver.
A projekt célja egy olyan webszerver kifejlesztése és karbantartása, amely biztonságos, hatékony és jól használható vállalati környezetben is. (*Apache HTTP Server*)

BSON (Binary JSON)

: A BSON felépítése megegyezik a JSON felépítésével, azonban itt nem csak a JSON specifikációban megadott adattípusok fordulhatnak elő, hanem egy BinData objektum, amelyben bináris fájlokat lehet tárolni.

Cache

: A cache használatával, a több ugyanolyan kérésnél nem szükséges minden alkalommal elvégezni a számítási műveleteket, mert a gyorsítótárból azonnal kiszolgálható az előző (a jelenlegivel megegyező) kérésre adott válasz eltárolt változatával.

CMS (Content Management System)

: Tartalomkezelő rendszer, melynek segítségével programozói tudás nélkül lehetséges a weboldalak készítése

CRUD (Create, Read, Update, Delete)

Az adattárolás négy alapvető műveletének (írás, olvasás, frissítés, törlés) összefoglaló neve.

ETag fejléc (header)

A weboldalaknál, kliensoldalon történő gyorsítótárazás egyik formája.

A webszerverről érkező válaszok verziózására használják, ugyanis amíg kérésre adott válasz nem változik, az ETag fejléc értéke sem fog, így a böngésző le sem tölti. Új verzió esetén az ETag fejléc változik, ezáltal jelezve a böngészőnek az új tartalmat.

Facebook

Az egyik legismertebb közösségi oldal. 2011 áprilisában több mint 500 millió felhasználóval rendelkezett. (Facebook statisztika, 2011)

FAT fájlrendszer

Eredetileg az MS-DOS fájlrendszere, később a Microsoft Windows is ezt használta. Az adattárolás során a nagy szektor méretek miatt, a fájlrendszer pazarlóan bánt merevlemezen tárolókapacitásával.

Geolokalizáció

Az internetre csatlakoztatott eszköz fizikai tartózkodási helyének meghatározása. Ez történhet a böngésző, vagy GPS segítségével.

Hash függvény

Olyan matematikai algoritmus, amely a bemeneti (tetszőleges hosszúságú) adatot véges hosszúságúra képezi le. Digitális aláírásoknál és ellenőrző szövegeknél is alkalmazzák.

HTML (HyperText Markup Language)

A HTML egy leírónyelv, mely minden weboldal alapja. A böngészők a HTML oldal struktúrája alapján jelenítik meg a honlapokat.

HTML5

A HTML új verziója, melybe már olyan megoldások kerültek bele, mint a geolokalizáció, az animációk támogatása, böngészőben futó adatbázisok, 3D modellezés.

HTTP kliens

A legtöbb programozási nyelvben megtalálható osztály, melynek segítségével HTTP kérések indíthatók.

JavaScript

A JavaScript egy objektumorientált és funkcionális programozási nyelv, mellyel leginkább a böngészőben találkozni. A nyelv egyszerűségével és könnyen tanulhatóságával, ma már sok keresztfordítóban (cross-compiler) megtalálható.

Load-balancer

Magyarul terhelés elosztó. Nagy terhelés esetén a kiszolgálás nem egy rendszerről történik, hanem többről, ilyenkor a terhelés elosztó dönti el (egy adott algoritmus alapján), hogy melyik rendszer szolgálja ki a bejövő kérést.

MD5 hash

Az MD5 hash 128 bites egyirányú kódolási algoritmus.

Microsoft SQL Server

Microsoft relációs adatbázis-kezelő rendszere, nagy népszerűségnek örvend, mivel fizetős alkalmazásokban is ingyenesen használható (az Express verzió).

MVCC (Multversion Concurrency Control)

Adatbázisoknál használt technika. Az adatbázis törlés esetén valójában nem töröl, csak törlésre jelöli az adott sort, illetve rekord frissítés esetén az adott rekord nem felülírásra kerül, hanem készül belőle egy újabb verzió.

Az MVCC-t használó rendszerek nagy előnye, hogy könnyen vissza lehet állítani az adatokat egy korábbi verzióra.

MySQL

Az egyik legismertebb relációs adatbázis-kezelő rendszer, 2010 óta az Oracle Corpotation tulajdona, egyszerű a használata és könnyen integrálható a legtöbb programozási nyelvvel

NoSQL (not only SQL) adatbázis

NoSQL adatbázisnak szokás nevezni, minden olyan adatbázist, amely

eltér a relációs adatbázisoktól (flexibilis sorméretek, JOIN műveletek elhagyása, jó skálázhatóság). A NoSQL rendszerek legismertebb típusa a dokumentum-orientált adatbázisok, így a kifejezéseket szinonimaként is szokás használni.

Oracle Database

Az Oracle Corporation egyik legismertebb terméke, relációs adatbázis-kezelő rendszer

PostgreSQL

Open-source relációs adatbázis-kezelő rendszer, a teljesen ingyenes használhatósága miatt közkedvelt mind az ingyenes, mind a fizetős alkalmazások között.

Reguláris kifejezés

A reguláris kifejezés segítségével, olyan szabályok hozhatók létre, melyekkel a szövegekben (a szabályok alapján) lehet keresni.

Replikáció

A replikáció során a rekordok egy másik adatbázis szerveren is megjelennek, ez lehet az adatbázis töredéke, vagy egésze is. Ha az egyik (master) adatbázisba történik rekord beillesztés, az automatikusan megjelenik a többiben is.

Az alkalmazások a replikációt a következőképpen szokták alkalmazni: az egyik adatbázisszervert kinevezik mester adatbázisnak, az adatok beszúrása ebbe történik, míg a többi kliens adatbázisként jelenik meg, azonban a kliensekbe nem történik rekord beszúrás, ezekből mindössze adatkiolvasás. Ezzel az adatbázis-konfigurációval csökkenthető a mester szerver terhelése, amely a költséges írási műveleteket végzi, miközben a gyakrabban történő kiolvasási műveletek a kevésbé leterhelt kliens adatbázisából történik.

RESTful (Representational State Transfer)

A kliens és a szerver közötti adatkommunikáció lebonyolításának egyik lehetséges technikája. A HTTP/1.1 szabvány által definiált lehetőségekre épít, azaz nem csak GET kérésekkel kommunikál, hanem GET (lekérdezés), POST (létrehozás), PUT(létrehozás, frissítés), DELETE (törlés) kérésekkel.

A weboldalakon gyakran alkalmazzák, ugyanis JavaScripttel végrehajtott AJAX (Asynchronous JavaScript and XML) kérésekben könnyebb az implementációja, mint más webszolgáltatásoknak (SOAP, RPC).

Sharding

A vertikális skálázás egyik formája, ilyenkor egy adatbázis szerver mesterként funkcionál, bár adatokat nem tárol, és a mester irányítja a shardingbe bevont kliens szerveket, melyek önállóan működnek, azonban egyik kliensen sincs meg a teljes adatbázis, mindegyik csak az információ egy részét tartalmazza.
A kliensek nem tudják, hogy ők shardingbe vannak kötve.

SOAP (webservice)

A kliens és a szerver közti adatkommunikáció lebonyolítására használt protokoll. Az üzenetek XML formátumban kerülnek küldésre, melynek szigorú felépítése van, amit egy WSDL (Web Services Description Language) állomány ír le.

SQL (Structured Query Language)

Általában a relációs adatbázis-kezelők által használt lekérdező nyelv. Megalkotása az 1970-es években történt az IBM-nél.

SQLite

Kisméretű relációs adatbázis-kezelő rendszer. Az SQLite adatbázisnak nincs szerverkomponense, az adatbázis mindössze egy fájlból áll.
A kis mérete és kompaktsága miatt szívesen alkalmazzák mind okostelefonokon, mind asztali alkalmazásokban (böngészők).

Twitter

Közösségi mikroblog szolgáltatás, segítségével a felhasználók maximum 140 karakterben tudnak egymásnak üzenni.
Az oldalon már közel 200 millió hozzáférést regisztráltak. (Twitter Hits Nearly 200M Accounts, 2011)

W3C (World Wide Consortium)

Egy konzorcium, amelynek a webes technológiák fejlesztésének irányítása, és technológiai iránymutatás (ajánlások) a célja.

XML (Extensible Markup Language)

Az XML egy leírónyelv, az adatok strukturált tárolására.

1. fejezet

Bevezetés

Az adatbázisok használata a számítógéppel egyidős, az adatok rendszerezése, karbantartása, kereshetősége a korábbi papíralapú rendszereknél komoly problémát okozott. A számítógép és internet alapú világban az adatbázisok használata mindennapossá vált, a számítógépeken, telefonokon futó alkalmazások legnagyobb része használ valamilyen adatbázist. Eddig az adatbázis fogalma alatt mindenki egy relációs rendszerre gondolt, amely sok problémára megoldást nyújt, azonban legalább ennyire csak szuboptimális. A dolgozat olyan rendszereket mutat be, amelyekkel megoldható a vertikális és a horizontális skálázhatóság, a cachelés, a memória optimális- és maximális kihasználása, a geolokalizációs információk natív indexelése és keresése, miközben az adatsorok mérete, és az oszlopok száma rugalmas. Mindez SQL tudás nélkül, megőrizve a relációs rendszerek legtöbb előnyét.

A dokumentum-orientált adatbázisokkal az alkalmazások fejlesztése nem csak felgyorsulhat, hanem olyan problémákra is megoldást nyújthat, melyek megoldása relációs rendszerekkel lehetetlen vagy nagyon nehéz. A rugalmas méretű adatsorok segítségével az adatbázis könnyedén felveszi az inkrementális fejlesztés támasztotta elvárásokat, és áthidalja a relációs adatbázisoknál gyakran felmerülő hézagos adatfeltöltés miatti átláthatatlanságot.
A memória optimális kihasználásával (akár egy teljes klaszter memóriáját képes kihasználni), az adatbázis-kezelő akár a relációs rendszerek válaszidejének töredéke alatt tudja kiszolgálni a kéréseket.
Az alapértelmezetten szállított vertikális skálázással (sharding), egy-egy új szerver bekapcsolása a szolgáltatásba lerövidülhet, ezáltal csökkentve az emberi erőforrásigényt, és növelve a szolgáltatásminőséget és folytonosságot.

2. fejezet

Adatbázisok történelme

Az adatbázisokat Gajdos Sándor a Budapesti Műszaki és Gazdaságtudományi Egyetem adjunktusa a következőképpen definiálja: *adatok gyors, gépesített tárolása és visszakeresése* (Gajdos, 2006, 4. old).

Az adatbázisok használatára már az elektromechanikus számológépek idejében felmerült az igény, igaz ekkor még csak lyukkártyára lyukasztották az információkat. Az adatbázisok térnyerése az 1960-as évekre tehető, amikor a számítógépek kizárólagos katonai szerepe megszűnt, mivel azok megfizethetővé váltak a cégek számára. Az ekkor használt rendszerek még az adatokat nem egy struktúraként kezelték, hanem az egyes rekordokra helyezték a hangsúlyt.

Az igazi áttörést, az IBM által az 1970-es években létrehozott SQL lekérdező nyelv érte el, amely az iparágban egy de facto szabványként terjedt el. Ekkor született meg a *relációs adatbázis-kezelő rendszer* kifejezés is.

Az 1970-es évek után a mai napig, az adatbázisok kifejezés általában relációs rendszerrel asszociálható. Természetesen, a relációs rendszerek mellett megjelentek más adatbázis rendszerek, melyek saját lekérdezőnyelvet használtak, azonban kiderült, hogy ezek sebessége és használhatósága nem veszi fel a versenyt a relációs társaikkal.

3. fejezet

Relációs adatbázisok

3.1. Kialakulás

A relációs adatbázis kifejezés Edgar Frank Codd[1] nevéhez fűződik, egy 1970-ben kiadott, a „A Relational Model of Data for Large Shared Data Banks" című cikkében használta először a fogalmat, és írta le 12 pontban mit ért a kifejezés alatt.

3.2. Jellemzők

A relációs adatbázis-kezelő szoftverek közül a vállalati szegmensben a legelterjedtebb az Oracle Database és a Microsoft SQL Server. (*Who Leads the RDBMS Pack?*)
A webes alkalmazásoknál előszeretettel alkalmazzák a MySQL és a PostgreSQL terméket, illetve a mobiltelefonokban az alacsony erőforrásigényéről ismert SQLite adatbázist. Az előző felsorolásból kitűnik, hogy a relációs adatbázisok felhasználási lehetőségei igen széleskörűek, hiszen egy bonyolult ERP rendszer, és a mobiltelefonon futó email kliens is ugyanarra az adatbázisrendszerre épít. A relációs adatbázisokban az információk táblákba rendeződnek, amelyeknek kötött formája van, mivel előre meg kell határozni milyen oszlopok fognak szerepelni, illetve az oszlopoknak meg kell adni az adattípusát (szöveg, egész szám, lebegőpontos szám, dátum, bináris adat, megjegyzés). Az adatok beviteléhez, és listázásához (legyen az akár egy grafikus felülettel ellátott adatbázis-kezelő alkalmazás) az SQL lekérdező nyelv használható, mellyel relációs műveletek segítségével kapható meg a megfelelő információ.

3.2.1. Előnyök

Relációk

Mivel a legtöbb adat felírható valamilyen relációs formában vagy legalább összekapcsolható egy relációs művelettel, ezért az adatbázisban szereplő rekordok kapcsolatban vannak egymással.

Konzisztencia

A táblák egymás közötti hivatkozásait és idegen kulcsait az adatbázis ellenőrzi, amelyből következően megakadályozza a rossz hivatkozások létrejöttét.

Például egy könyvtári rendszer adatbázisában, olyan könyv nem törölhető, melyre van hivatkozás a jelenleg kint lévő könyveket tartalmazó táblában.

[1] Brit matematikus, informatikus, az IBM kutatójaként dolgozott

Indexelés

Az adatbázisban történő keresést az indexelés jelentősen felgyorsítja, mivel az index fájlban eltárolásra kerülnek az indexelt oszlopban található adatra történő hivatkozások, és a keresés az index fájlban történik.

Redundancia megszüntetése

Minden relációs rendszer a redundanciamentességre, azaz az adatok többszöri jelenlétének elkerülésére törekszik.

Tranzakciók

A tranzakciók segítségével egy komplex adatbeillesztés (több lépésből álló adatbeszúrások) hibája esetén, felmerülő félig beillesztett rekordok elkerülhetőek, ugyanis hiba esetén a tranzakció visszavonható.
A tranzakciók a pénzügyi rendszerek egyik alapjai.

3.2.2. Hátrányok

Rugalmatlanság

Egy relációs adatbázis kevésbé flexibilis, mert nehezen követi az inkrementális fejlesztés változó követelményeit, problémákat okozhat a verziókövetés, a visszafelé kompatibilitás. További problémák lehetnek például:

- a különböző tulajdonságokkal rendelkező entitások leírása:
  például egy címjegyzék, ahol a kapcsolatok nagy részének egy vagy kettő telefonszáma van, de előfordulhat a három telefonszám, email cím, beosztás és még az egyes kontaktokhoz létrehozott egyéni mezők kombinációja is.
- a hézagos adatfeltöltések: például az adóbevallás, ahol mindenkinél szerepel minden kitölthető mező, viszont a kitöltők mindössze a mezők töredék részét töltik ki.

Relációk

A legtöbb információ leírható relációkkal, azonban egy közösségi oldalnál, ahol a felhasználók rengeteg mezőt kitölthetnek (név, lakcím, iskolák, email címek, stb.), és még saját mezőket is felvehetnek, gyakran a relációs megközelítés szuboptimális, a nagy adathalmazon mérhető keresési idők miatt. A problémát jól szemléltetheti egy gyógyszeradatbázis, ahol ki kell tölteni minden gyógyszer

alapanyagát, amely viszont sok oszlopot eredményez a táblákban - hézagos kitöltés és *NULL* értékek megadása a relációkat felboríthatja.

Gyakran a sok relációval leírt adatok lekérdezése időigényes lehet, illetve sok információt nehéz leírni relációval, vagy az adatbázist feleslegesen bonyolulttá teszi. Megfigyelhető, hogy a komolyabb adattárházakban nem minden adatbázis a lehető legoptimálisabb, mivel dönteni kell a teljesítmény és az áttekinthetőség között.

Fregmentáció

Az adatbázis bináris fájlként kerül tárolásra a lemezen. Ha az adatbázis séma módosul egy új oszlop hozzáadásával, azt az adatbázis-kezelő rendszer már nem tudja a lemezen a korábban felvett oszlopok mellé menteni, hanem a következő szabad részre kerül, így az adott táblából történő olvasáskor a diszk olvasófeje folyamatosan ugrálni fog, a válaszidők meghosszabbodását okozva. Hasonlóan problémát jelenthet egy oszlop törlése, mivel az adatbázis fizikai méretének csökkenésével nem jár. Az adatbázis fájlszerkezetét leginkább a FAT fájlrendszerhez lehetne hasonlítani, ahol a fregmentáció miatt gyakran szükséges töredezettségmentesítést futtatni.

Ez a probléma nem érinti az adatbázisban tárolt megjegyzéseket és bináris állományokat, mert azok az adatbázisrendszer külön tárolja (az oszlop méret flexibilis mivolta miatt).

Hosszú válaszidők

A nagy adatbázisoknál gyakori probléma, hogy a sok relációs kapcsolat hosszú válaszidőket okoz (soktáblás JOIN műveletek).

Bináris fájltárolási problémák

A legtöbb relációs adatbázis támogatja a fájlok tárolását, azonban egy-egy fájllekérés hatalmas plusz terhelést ad a szervernek, így általában a gyakorlatban nem alkalmazzák.

Memória (optimális) kihasználása

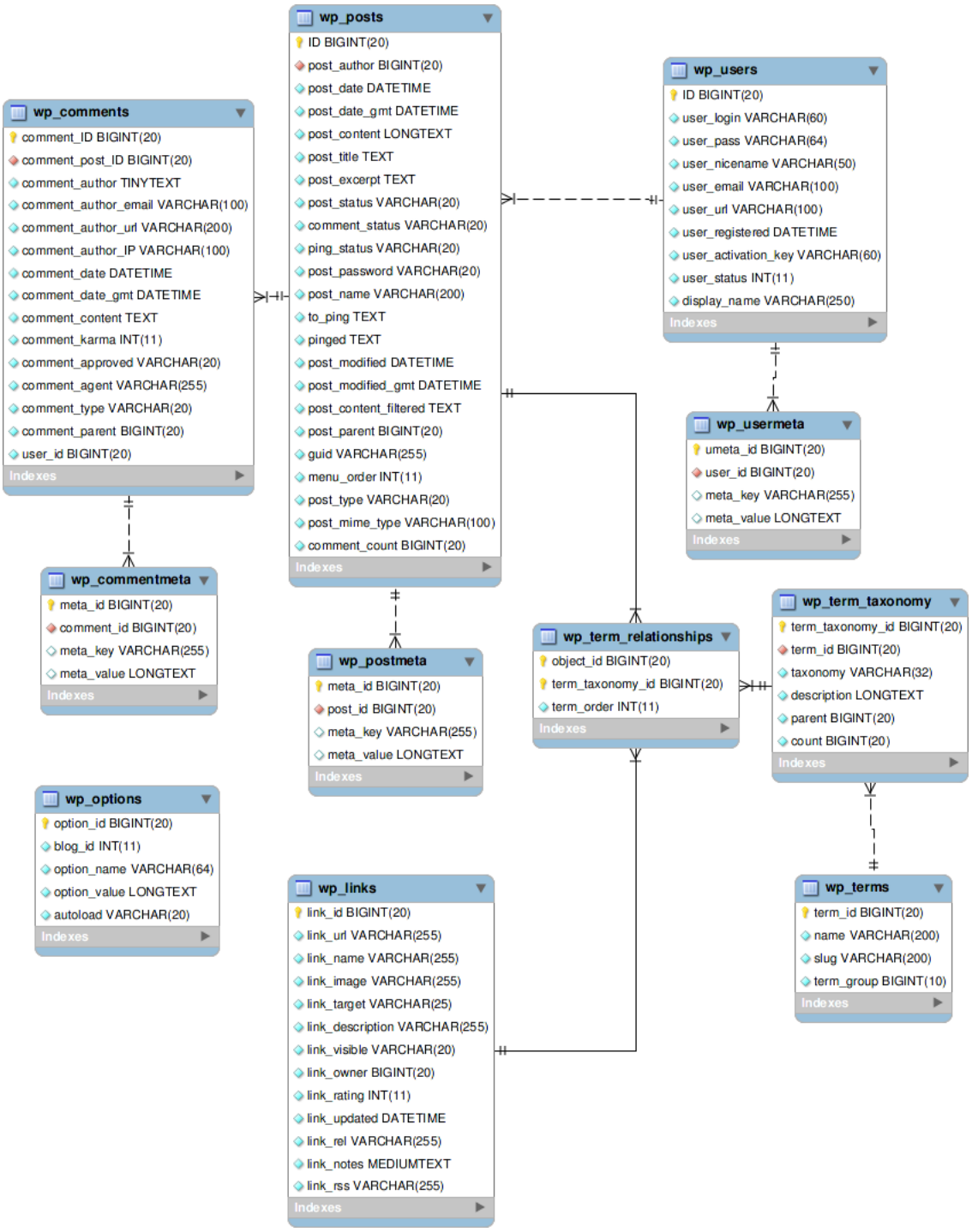
Az egyes adatbázisrendszerek nem, vagy nem optimálisan használják ki a memória által nyújtott cachelési lehetőségeket. Ennek egyik oka, hogy a 32 bites architektúrákban a memória maximum 3 gigabájtig címezhető, így az adatbázis-kezelő mindig a lemezre írja, és a lemezről olvassa az információkat, hosszú válaszidőket eredményezve.

3.3. ORM

Object-relational mapping

Az ORM egy olyan programozási technika, melynek segítségével egy relációs adatbázist objektumokká lehet konvertálni.

Miért van erre szükség? A ma használt programozási nyelvek legnagyobb része objektumorientált, azaz az adatbázisból érkező lekérdezések eredményét a memóriában objektumok formájában kell felépíteni.
A nagy adatbázisoknál a nagy adathalmaz miatt komoly problémát okoz a relációk felépítése a táblák között. A 3.3 ábra ezt kívánja szemléltetni.



3.1. ábra. A népszerű Wordpress CMS adatbázisának felépítése

A Wordpress, az egyik legismertebb PHP alapú tartalomkezelő rendszer. 2011. áprilisában közel 20 millió weboldalnál és blognál alkalmazták.

Az adatbázisának felépítése egyszerűnek mondható, bár itt is problémát okozhat a különböző táblák és a táblákban szereplő oszlopok fejben tartása. Egy ERP rendszernél, ahol több száz tábla is lehet, táblánként több tíz oszloppal az átláthatóság szinte lehetetlen.

A ORM használatakor a programozási nyelv SQL mondatokká alakítja a kérést, majd az adatbázis által visszaadott választ a memóriában objektumokból felépülő listává alakítja, később az adatbázis írásakor az objektumokban történt változásokat visszaalakítja SQL mondatokká.
Fontos megemlíteni az ilyenkor használt két technikát, az optimista és a pesszimista zárolást.

Optimista zárolás Az optimista zárolás esetén a lekérdezés lefut, az adatok memóriába kerülnek, majd visszaíráskor dönti el az adatbázis időbélyegzők (timestamp) alapján, hogy frissült-e az adatbázis. Tehát míg az egyik kliens módosítja az információkat addig mások hozzáférhetnek az adatokhoz és akár ők is módosítják. Ez a megvalósítás akkor lehetséges, ha az alkalmazás használata közben feltételezhető, hogy két felhasználó egyszerre nem akarja módosítani az ugyanazt a rekordot, vagy nagyon kicsi az esélye az egyidejű adatmódosításnak.

Pesszimista zárolás A pesszimista zárolás esetén, ha egy kliens szerkesztésre megnyit az adatbázisban szereplő információkat, akkor az adott rekordok zárolásra kerülnek, majd a módosítás elküldése után válik lehetővé a többi kliens számára is a módosítás, tehát ha egy kliens szerkeszti az információkat, a többiek csak megtekinteni tudják azt, a módosítás nem lehetséges.
Ennél a megoldásánál problémát jelenthet a kliens leszakadása a hálózatról, ugyanis ilyenkor nem egyértelmű, hogy a zárolást mikor kell feloldani.

Az, hogy optimista vagy pesszimista zárolást használjon egy alkalmazás, erősen függ a tárolt adatoktól. Például egy pénzügyi rendszer valószínűleg a pesszimista módszert fogja választani, míg egy egyszerű blognak megfelelő választás az optimista is.

4. fejezet

NoSQL adatbázisok

Az új szemlélet

4.1. Bevezetés

Az előző fejezetben bemutatásra került relációs adatbázisok sok pozitív tulajdonsággal rendelkeznek, ezért sok helyen és formában alkalmazzák őket, mivel a legtöbb a probléma megoldásához kiválóan használhatóak. Azonban, mint a legtöbb rendszernél itt is szükséges kompromisszumokat kötni, hiszen rendelkeznek hátrányokkal is (3.2.2). A problémák nem érintenek minden alkalmazást, azonban vannak olyan alkalmazások, ahol nem egy relációs adatbázis a tökéletes megoldás. Az ilyen problémák megoldására jöttek létre a NoSQL adatbázisok.

Az első komolyabb adatbázis, amely szakított a klasszikus relációs megvalósítási formával, az 1989-ben megjelent Lotus Notes[1] szerver oldali komponense a Lotus Domino volt. Természetesen már korábban is voltak próbálkozások, egészen pontosan az 1980-as években kifejlesztett BerkleyDB, vagy a szintén ekkor készült GT.M megoldás, de ezek nem tudtak elterjedni, illetve említésre méltó piaci részesedést elérni, ugyanis az 1980-as években sem az (adatbázist használó) alkalmazások száma nem volt magas, sem az alkalmazások nem igényeltek rugalmas adattárolási technikákat.

A következő említésre méltó adatbázis-kezelő az 1997-ben elkészült Metakit volt, melyet az első dokumentum-orientált adatbázisnak is tartanak. A termék sikerességét mutatja, hogy az Apple Mac OS X termékében ezt használták a címjegyzék adatbázisának. A Metakit legutóbbi verziója 2007-ben jelent meg és a fejlesztői levelezőlistája még a mai napig aktív.

A „dotkomlufi" kipukkanása a NoSQL adatbázisoknak is hatalmas lökést adott, melyet a következő felsorolás szemléltet:

Memcached (2003)

> *Key-value (kulcs-érték)*, bárki számára ingyenesen használható adatbázis, amely a memóriában tárolja az adatokat, így biztosítva a rövid válaszidőket. Számos projektben használják átmeneti tárolóegységként (cache), ezáltal csökkentve az adatbázisrendszer terhelését, és az alkalmazás erőforrásigényét. Mivel a Memcached csak kulcs-érték adatbázis, így egyszerű adatstruktúrát érdemes benne tárolni. Szerializált szövegek használatával bonyolultabb struktúrák elhelyezése is megoldható, azonban ekkor le kell mondani a keresésről, és az indexelhetőségről.

[1] Ma már IBM Lotus Notes, ugyanis 1995-ben az IBM megvásárolta Lotus céget

Google BigTable (2004)

A Google saját fejlesztésű teljesen zárt adatbázisa, külső fejlesztőknek számára nem érhető el. Ma már a legtöbb Google termék ezt használja (Reader, Youtube, Mail, Maps).

Apache CouchDB (2005)

Az *Apache Software Foundation* termékét a Lotus Notes ihlette adatbázis alapjaira fejlesztették, az igazi elterjedése az elmúlt két évben indult meg. A CouchDB-vel a 4.3 fejezet foglalkozik bővebben.

A következő időszakot, amelyet körülbelül 2005-től számítanak, a webes alkalmazások folyamatos térnyerése idézte elő. Ide tartozik a Google e-mail szolgáltatása, a GMail, a Twitter, és a Facebook. Ekkor (2007) jelent meg, a mára komoly hírnevet kivívó adatbázis, a MongoDB, mely a későbbiekben részletesen bemutatásra kerül a 4.4 fejezetben. 2008-ban pedig a Facebook által nyílt forráskódúvá (open-source) tett Cassandra projekt vált elérhető a fejlesztők számára, amelynek fejlesztéséért szintén az Apache Software Foundation a felelős. A Cassandra-t kimagaslóan jó skálázhatóságának köszönhetően ma már a Facebook, és a Twitter is használja. A Cassandra skálázása és replikációja mögött a peer-to-peer technológia áll, ezáltal növelve az adatbázis rendelkezésre állási idejét.
2010-ben jelent meg a Membase, a Memcached klaszterekre szánt verziója, melynek segítségével akár egy több száz gépből álló szerverpark memóriája válik használhatóvá gyorsítótárazásra.

4.2. Miben más?

A NoSQL adatbázisok létrejötte az *új típusú* alkalmazások megjelenésének köszönhető. De mit is jelent az új típusú? A következő egyszerű példán keresztül kívánom bemutatni:
Adott szervezet honlapján az ügyfeleknek egy regisztrációs űrlapot kell kitölteniük, amelyen az ügyfél legfontosabb adatai (név, lakcím, születési dátum, fénykép, beszélt nyelvek) szerepelnek. Ezen adatok a szerveroldalon egy relációs adatbázisba kerülnek elmentésre. Ezzel idáig nincs is semmi probléma, az ügyfelek kitöltik, a cég pedig tud keresni, szűrni a kitöltött űrlapok között, illetve meg tudja tekinteni őket.
De mi történik, ha fel kell venni a mezők közé még egy plusz információt (tele-

fonszám, Facebook-profil)? Amennyiben nincs erre felkészítve a rendszer, bár ha fel is van, nem lesz tökéletes az adatbázis-struktúra, ugyanis ebben az esetben a különböző mezők típusa vagy mindig szöveg, vagy rengeteg felesleges oszlop kerül az adatbázisba, hogy minden adat a neki megfelelő típusban legyen elmentve (szöveg, dátum, fájl). Tehát ha nincs felkészítve az adatbázis erre, akkor fel kell venni a kapcsolatot a honlapot készítő vállalattal, akiknek ki kell ezt javítani a honlapon, a szerver-, a kliensoldalon és az adatbázisban is, amely arányaiban véve nagyon sok idő. Ehhez a munkához szükség van legalább az adatbázisért felelős személyre, egy kliens- és egy szerveroldali programozóra.
Azonban mi történik, ha egy NoSQL adatbázis dolgozik a háttérben? Az eredeti feladat megoldása itt sem okoz komolyabb problémát, viszont ha ki kell egészíteni azzal a bizonyos plusz mezővel, könnyedén megtehető (az adatbázis módosítása nélkül), ugyanis nincs megkötve, hogy hány oszlopból kell állnia egy sornak, illetve milyen adattípus szerepelhet az adott oszlopokban (gyenge típusosság).

Ezek után könnyen levonatható a tanulság, hogy milyen rendszerekben alkalmazható jól egy NoSQL adatbázis: folyamatosan módosuló adatszerkezeteknél, a felhasználók által testreszabható megoldásoknál, illetve az eltérő entitásokkal rendelkező adatstruktúráknál.
Bár a dokumentum-orientált adatbázisok eme rugalmassága kényelmes megoldásnak tűnik a relációs adatbázisokhoz képest, azonban ezt már a legkorábbi key-value adatbázisok is könnyedén meg tudták oldani. A NoSQL adatbázisok valódi sikerének kulcsa azon a problémák a megoldásában rejtezik, melyet a mai modern alkalmazásfejlesztési problémák hoztak a felszínre:

Teljes szöveges keresés (full-text search)

A teljes szöveges keresés lehetősége a legtöbb relációs adatbázisban megtalálható, azonban a fejlesztők ahol lehet, megpróbálják az adatkeresésnek ezt a módját elkerülni lassúsága miatt.

Az SQL nyelv

Az SQL nyelv bármennyire is beszélt nyelv közeli, egy komolyabb méretű adatbázisnál problémákat okozhat (erre jelenthet megoldást a 3.3 fejezetben kifejtett ORM).

Problémák forrása lehet, a tíz tábla feletti, szűréssel és aggregációval ellátott lekérdezések megírása és átlátása.

SQL-ben nem jellemző, hogy a fejlesztők kommentekkel egészítenék ki az SQL mondatokat, amely szintén megértési problémát okozhat.

Geolokalizáció

Ma már sok alkalmazás használ geolokalizációt, például a legtöbb közösségi hálózat (Facebook, Twitter), vagy éppen a webes térképszoftverek (Bing Maps, Google Maps).

Bináris fájlok tárolása

A legtöbb NoSQL adatbázist fájltárolási támogatással szállítják, amely szakít a relációs rendszerek bináris állományok tárolásának technikájával. A fájlok strukturáltan tárolódnak, általában szétdarabolva, a gyors elérés érdekében.

Horizontális, vertikális skálázhatóság

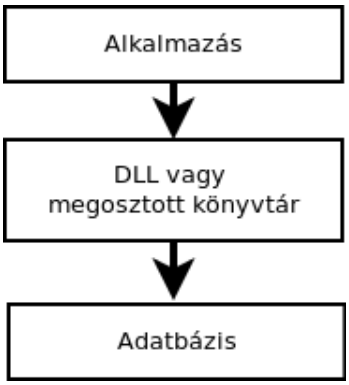
A horizontális skálázás a legtöbb relációs rendszerben megtalálható, ezt nevezik replikációnak, ilyenkor a kliens (*slave*) vár a mester (*master*) szerver üzenetére, hogy milyen változás történt az adatbázisban, amelyet a kliens is végrehajt. A vertikális skálázás is általában replikációval történik, azonban ez nem kliens-mester replikáció, hanem mester-mester, ahol minden szerver kliensként és mesterként is rész vesz a kommunikációban.

4.3. Apache CouchDB

A CouchDB az Apache webszerver fejlesztéséért is felelős Apache Software Foundation terméke. Az első verzió 2005-ben jelent meg. Az adatbázis felépítése erősen hasonlít a IBM Lotus Notes termékére, ugyanis a projekt vezető fejlesztője, Damien Katz, korábban a Lotus Notes fejlesztésében is részt vett.
Az információk tárolása adatbázisokban és dokumentumokban történik. Az adatok eléréséhez nincs szükség SQL tudásra, ugyanis JavaScript segítségével végezhetők el a szűrések -amelyet nézetnek (view-nak) neveznek, egészen pontosan a MapReduce használatával.

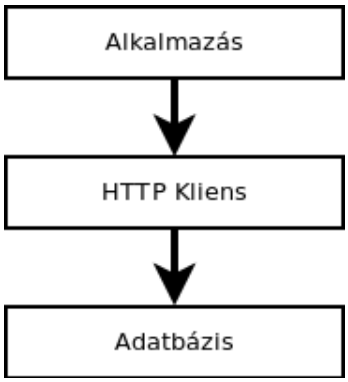
4.3.1. Interfész

A CouchDB felépítése eltér a klasszikus adatbázisoktól. Az 4.1 ábrán egy olyan alkalmazás általános kapcsolódási interfésze látható, ahol az alkalmazás natív driverekkel (DLL fájl, vagy megosztott könyvtár) kapcsolódik az adatbázishoz.



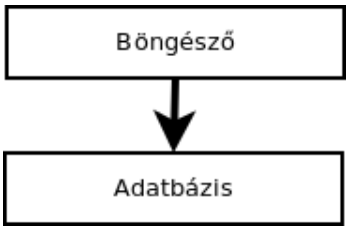
4.1. ábra. Példa egy általános kapcsolódási interfészre

A 4.2 ábrán látható, ahogyan egy alkalmazás csatlakozhat a CouchDB-hez, itt azonban nincs szükség, semmilyen illesztőre, mindössze egy HTTP kliensre. Ennek az az előnye, hogy bármilyen programozási nyelvből indítható kérés az adatbázis felé. Hátránya viszont, hogy a lekérések nem lesznek olyan gyorsak, mint egy natív driver segítségével.



4.2. ábra. Kapcsolódás CouchDB-hez szerver oldali rétegen keresztül

A 4.3 ábrán látható kapcsolódás már jobban megmagyarázza, miért jó a HTTP interfész. Ha egy weboldalt veszünk példának, akkor a kliens maga a böngésző lesz és szerveroldali programozási nyelv használata nélkül lehetségessé válik az információk megjelenítése.



4.3. ábra. A kliens közvetlen kapcsolódása az adatbázishoz

Miért jó ez?
Az egyszerű honlapoknál, a szerveroldali rész (legyen az PHP, ASP.NET vagy Perl) feladata mindössze annyi, hogy kapcsolódik az adatbázishoz, majd a lekérdezés eredménye alapján, felépíti a megfelelő HTML oldalt. Ha lekerül HTML tartalom összeállításának terhe a szerverről, tehát kliensoldalon kerül felépítésre a tartalom, rengeteg erőforrás spórolható meg, ezáltal költséghatékonyabb szerver-üzemeltetés érhető el, így tulajdonképpen az ügyfél erőforrásaival fizet a szolgáltatás használatáért.
Azonban ennek a technikának is van hátránya, mivel ami a weboldalaknál kliensoldalon (böngészőben) történik az bárki számára látható, ezért itt kiemelt szerepet kap a biztonság. Szerencsére a CouchDB nyújt lehetőséget a különböző adatbázisok jelszóval történő védelmére, illetve kívülről láthatatlanná tételére. Ilyenkor fontos, hogy csak azok az adatbázisok legyenek megtekinthetőek (kliensoldalról elérhetőek), amelyek nem tartalmaznak érzékeny információkat, például a regisztrált felhasználók vagy az adminisztrátorok személyes adatait.

A HTTP interfész

A CouchDB HTTP interfésze egy RESTful webszolgáltatás, mely jelentősen eltér a webszolgáltatások által korábban preferált SOAP technikától. Itt ugyanis nem egy XML fájl segítségével történik a webszerver utasítása, hanem standard HTTP kérésekkel. Dokumentumok lekérdezése GET-, egy adott dokumentum lekérdezése (az azonosítója megadásával) szintén GET kéréssel történik. A létrehozás POST, a frissítés PUT (azonosító megadása szükséges), míg a törlés DELETE (azonosító megadása szükséges) kérés küldésével hajtható végre. Megfigyelhető, hogy a kérések mennyire jól tükrözik a CRUD műveleteket.

Miért HTTP, és nem natív interfész? A HTTP interfész nagy előnye, hogy ugyanazok az eszközök használhatók a performancia javítása érdekében,

mint a webes alapú rendszereknél: terhelés elosztás (load-balancer), gyorsítótárak (cache), SSL- és autentikációs proxy. Például a dokumentumok revíziós száma kiválóan használható ETag fejlécként a cache szerveren.

4.3.2. Az adatmodell

Dokumentum (Document)

A CouchDB adatstruktúrájának legfontosabb része a dokumentum, amely megközelítőleg a relációs adatbázisok megadott táblája egy sorának feleltethető meg. Egy dokumentumban mezők találhatóak, melyek kulcs-érték formában jelennek meg. A kulcs bármilyen szöveg lehet, míg az érték a következők egyike:

- szöveg (string)
- egész szám (integer)
- lebegőpontos szám (float)
- dátum (JavaScript Date)
- objektumok (JavaScript object, Array)

Továbbá tartalmazhat hivatkozásokat más dokumentumra (URL-k), azonban nem kerül ellenőrzése a hivatkozás, illetve az adatbázis nem tartja fenn a konzisztenciát. A dokumentumok egymásba nem ágyazhatók.

A 4.4 ábrán látható a dolgozat egy részlete CouchDB adatbázisban tárolva. A dolgozat címe, készítési ideje, és tartalma hasonlóan van tárolva, mint egy relációs rendszerben, azonban megfigyelhető, hogy míg egy relációs adatbázisban a címkékhez és a kommentekhez biztosan szükséges lenne egy kereszttábla beiktatása, itt egyszerűen egy tömbbel kifejezhető az egy a többes kapcsolatot.
A dokumentum azonosítója ("_id" mező) egy 128 bites, automatikusan generált érték (tehát a CouchDB egy adatbázisban maximum 3.4^{38} dokumentumot tud tárolni). A revíziós szám is egy generált érték, azonban ez csak 32 bites, és egy hash függvény állítja elő. A revíziós szám használatával a dokumentum minden frissítésekor fizikailag egy új dokumentum jön létre. A lekérdezésekben a dokumentumnak mindig a legutolsó verziója látható. Tehát a revíziós szám, csak a dokumentumok

különböző verzióinak megkülönböztetésére szolgál, illetve ennek segítségével oldja meg a tranzakciókat illetve replikációt is. (Anderson et. al., 2010a)

```
{
    "_id": "a040e00418996ae96e91ef998f000eea",
    "_rev": "1-3fab44526d5d7d3a52ab1761b323ab62",
    "cím": "Dokumentum-orientált adatbázisok",
    "címkék": ["nosql", "couchdb", "skálázhatóság", "mapreduce"],
    "létrehozva": "Wed Apr 20 2011 11:45:15 GMT+0200 (CEST)",
    "tartalom": "CouchDB az Apache Foundation terméke...",
    "kommentek": [{
        "felhasználó": {
            "név": "felhasználó1",
            "ip": "12.12.12.12"
        },
        "üzenet": "CouchDB - a dokumentum-orientált adatbázis."
    }],
    "_revs_info": [{
        "rev": "1-3fab44526d5d7d3a52ab1761b323ab62",
        "status": "available"
    }]
}
```

4.4. ábra. Egy egyszerű CouchDB dokumentum

A dokumentumok szerializált JSON adatként kerülnek a lemezre.

Nézet (View)

CouchDB-ben minden lekérdezést, csoportosítást, sorba rendezést nézetnek neveznek.

A nézetek JavaScriptben írt függvények, amelyek nem tudják módosítani az adatbázisban tárolt adatokat, csak a elvégzik a szűrést (vagy csoportosítást, vagy éppen a sorba rendezést).
A JavaScript függvények MapReduce alapúak, tehát minden függvénynek létezik egy *map* és egy *reduce* része.

```
function ( document ) {
    if ( document[ "cím" ] === "Dokumentum-orientált adatbázisok" ) {
        emit(null, {
            "cím": document[ "cím" ],
            "címkék": document[ "címkék" ],
            "létrehozva": document[ "létrehozva" ],
            "tartalom": document[ "tartalom" ]
            "kommentek": document[ "kommentek" ]
        });
    }
}
```

4.5. ábra. A CouchDB Map függvénye

Map. A map függvénynek egy paramétere van, amely egy dokumentum, a függvényben itt végezhető az adott dokumentumra vonatkozó szűrés. A 4.5 ábrán látható a CouchDB map függvénye használat közben, ahol a dokumentum címére történik a szűrés, tehát kijelenthető, hogy a map megfelel a relációs adatbázis szűrés (WHERE) záradékának.

Amennyiben a cím megfelel a feltételnek, akkor átadásra kerül reduce függvénynek az *emit* használatával. (Introduction to CouchDB Views, 2011)

```
function (document) {
    document[ "kommentek" ] &&
        document[ "kommentek" ].forEach(function ( komment ) {
            emit( komment[ "felhasználó" ][ "név" ], 1 );
        });
}

function ( nev, kommentSzam ){
    return {
            "felhasználónév": nev,
            "összes": sum( kommentSzam )
           }
}
```

4.6. ábra. A CouchDB map és reduce függvénye

Reduce. A Reduce függvény alkalmazása opcionális, segítségével aggregálható a map által szűrt információ. A 4.6 kódrészletben, az előző bekezdésben megismert map függvény látható más formában. Első paraméter itt a kommentelő neve lesz, a második pedig az egy, ezt megkapja a reduce és összesíti az egy névhez tartozó egyeseket, majd függvény visszaadja melyik kommentelő hány hozzászólást írt eddig.

4.3.3. Replikáció

A CouchDB replikációja hasonlóan működik, mint a relációs adatbázis rendszereké, lehetővé teszi az adatok megosztását az adatbázisok között.
Viszont a replikáció indításához itt nincs szükség különböző konfigurációs fájlok írására, ugyanis egy POST kéréssel indítható illetve megszakítható (specifikálni kell a kérésben a forrás és a cél adatbázist), amely lehet egyszeri vagy folyamatos szinkronizáció. (CouchDB Replication, 2010)
A replikáció indításkor az adatbázisok összehasonlításra kerülnek, majd a hiányzó adatok átkerülnek a céladatbázisba. A folyamat során csak azok a dokumentumok és mezők kerülnek át, melyek frissültek (a 4.3.2 fejezetben olvasható revíziós szám ebben nyújt segítséget). A szinkronizáció leállása (hálózati, vagy szoftveres hiba) esetén a replikáció attól a dokumentumtól folytatódik, ahol a leállás történt.

4.3.4. Tranzakciók

A CouchDB támogatja a tranzakciókat. A tranzakciók a 4.3.1 fejezetben olvasható revíziós szám alapján történik. Azonban a CouchDB-ben található tranzakció nem egyezik a relációs adatbázisokban használttal. Itt ugyanis a tranzakciót, nem egy több lépésből álló műveletként kezelik, hanem az egyes műveletek számítanak egy tranzakciónak.
Például, ha egy banki rendszerben átutalás történik, akkor az adott ügyfél számlájáról először leemelésre kerül az összeg, átkerül a folyamatban lévő átutalások közé, majd pedig arra a bankszámlára, amelyre a pénzt küldték. Ha ebben a folyamatban valami probléma történik (nincs elég pénz a számlán, nem lehetséges elindítani az utalást), akkor a tranzakció visszavonásával, minden visszaáll az eredeti álla-

potba. A CouchDB-ben ez azonban máshogy történik, mivel minden egyes lépés (a pénz leemelése, áthelyezése, elküldése) külön tranzakcióként hajtódik végre, ezért nem lehet egyben visszavonni hiba esetén. Viszont a tranzakciók egyesével, egy ellenirányú tranzakció létrehozásával visszavonhatóak. (Anderson et. al., 2010b)

Bár a CouchDB már 2005 óta elérhető a nagyközönség számára, a valódi felfutása 2009 és 2010 közt indult meg. Azóta már olyan cégek használják mint a svájci székhelyű Nukleáris Kutatások Tanácsa, a CERN[2], ahol évente az adatbázisba körülbelül 10 petabájt információ kerül be. (CERN, 2010)
A CouchDB már rendelkezik saját hoszting szolgáltatóval, ez a CouchBase, amely a CouchDB-t ötvözte a Membase szolgáltatásaival, ezáltal csökkentve válaszidőket, az adatbázis terhelést és növelve a szolgáltatásfolytonosságot A szolgáltatás minőségét jól jellemzi, hogy a CERN mellett már nagy cégek is használják, mint a BBC, a PayPal, a Adobe és a Vodafone.

4.4. MongoDB

A MongoDB egy dokumentum-orientált adatbázis, a 10gen Inc. cég terméke. A fejlesztés célja az volt, hogy egy skálázható kulcs-érték adatbázist hozzanak létre, amely kiválóan megoldja, a gyakran felmerülő problémákat az alkalmazások fejlesztésében.
A MongoDB-t egyre szívesebben választják komoly felhasználói bázissal rendelkező szolgáltatások, amelyek közül vannak, akik a korábbi relációs adatbázis rendszereiket teljesen lecserélik, vannak akik párhuzamosan használják a két rendszert, kiaknázva az erősségeiket. 2011-re olyan cégek döntöttek a MongoDB mellett, mint az open-source szoftverek gyűjtőhelye a Sourceforge.net, a foursquare közösségi oldal, vagy éppen a New York Times.

4.4.1. Interfész

A MongoDB kapcsolódási interfésze megegyezik a relációs adatbázisoknál használt technikával, azaz natív driverekkel lehet kapcsolódni, bár szükség esetén kiegészíthető RESTful webszolgáltatás bővítménnyel.

[2]CERN (Conseil Européenne pour la Recherche Nucléaire): a Nukleáris Kutatások Európai Tanácsa

Az adatbázishoz vagy valamilyen támogatott programozási nyelv, vagy a szerverkomponens mellé települő konzolos kliens segítségével lehet hozzáférni. A MongoDB-hez hivatalosan támogatott grafikus felülettel ellátott adatbázis-kezelő nem készült még, azonban a felhasználók által készített kliensprogramok között már találunk ilyet. Ezek minőségével és használhatóságával vannak problémák, így a következő példák is konzol segítségével készültek.

A konzol elérhető a MongoDB által támogatott platformokon (Windows, OS X, Linux, Solaris), Windows operációs rendszeren a parancssorba, míg Linuxon a terminálba beírva a *mongo* parancsot.

A konzolban néhány kivételtől eltekintve (adatbázis létrehozás, listázás) az utasítások JavaScript függvények.

A 4.7 ábrán látható egy adatbázis létrehozása, ahol mindössze a *use <név>* utasítás kiadásával, már létre is jött az adatbázis.

```
1 use újAdatbázis;
```

4.7. ábra. Adatbázis létrehozása MongoDB-ben

A 4.7 ábrán látható kifejezés lefuttatása után, az adatbázis a *db* változó segítségével férhető hozzá, ezáltal megkönnyítve a programozó dolgát, a lekérdezés megírásakor az adatbázis nevének begépelésétől.

A gyűjtemény létrehozása sem sokkal bonyolultabb, mint az látható a 4.8 ábrán, a *createCollection(<név>, <opcionális paraméterek>)* parancs kiadásával jön létre az adott adatbázisban.

```
1 db.createCollection( "újGyűjtemény" );
```

4.8. ábra. Gyűjtemény létrehozása MongoDB-ben

4.4.2. Az adatmodell

A MongoDB felépítése hasonlít a relációs adatbázisok adatszerkezetére, vannak adatbázisok, amelyek gyűjteményeket (Collections) tartalmaznak. A gyűjtemények megfeleltethetőek a relációs adatbázisok tábláinak, amelyek tartalmazzák az adatsorokat. Egy adatbázis több gyűjteménnyel is rendelkezhet. Nagy különbség a többi adatbázishoz képest, hogy ha itt egy új sor kerül beszúrásra, és az adatbázis vagy a gyűjtemény nem létezik még, akkor automatikusan létrehozásra kerül, a konfigurációban megadott paraméterekkel, sőt alapértelmezetten automatikus indexeléssel.
A gyűjtemények dokumentumokból épülnek fel.

Dokumentum

MongoDB-ben a dokumentum hasonló szerepet tölt be mint a CouchDB-nél (4.3.2), itt is az adatmodell legalsó rétege. A 4.9 ábrán látható egy dokumentum. Megfigyelhető, hogy az adat itt is JSON formátumban jelenik meg, valójában ez azonban BSON, ugyanis az adatbázis a bináris fájlokat is ebben a struktúrában tárolja, és a JSON formátum specifikációja (ECMAScript, 2009) nem ad lehetőséget bináris adatok tárolására.
A dokumentumok mérete maximum 4 kilobájt lehet.

```
{
    "_id": ObjectId("4daaf1ada115ac88de83f8f1"),
    "dbTípus": "MongoDB",
    "relációs": false,
    "tulajdonságok": [
            "skálázható",
            "gyors",
            "optimalizált memóriakihasználás"
            ],
    "létrehozva": " Wed Apr 20 2011 17:45:15 GMT +0200 ( CEST )" ,

}
```

4.9. ábra. Egy egyszerű MongoDB dokumentum

```
db.újGyűjtemeny.insert({
    "újÉrtek": true,
    "létrehozva": ( new Date() ),
    "név": "MongoDB"
});
```

4.10. ábra. Dokumentum létrehozása MongoDB-ben

Egy dokumentum létrehozása látható a 4.10 ábrán.
A MongoDB dokumentumokban a következő adattípusok tárolhatók:

- igaz/hamis értékek (boolean)
- egész számok (integer)
- lebegőpontos számok (double)
- szöveg (string)
- dátum (JavaScript Date)
- reguláris kifejezések (JavaScript RegExp-ek)
- objektumok (JavaScript object, Array)
- üres értékek (null)
- id (ObjectID)

Lekérdezések

A MongoDB-ben történő lekérdezések alapja is a MapReduce, akárcsak a CouchDB-nél (4.3.2). Itt azonban a CouchDB-vel ellentétben, nem kell megírni minden egyes szűrést vagy keresést végző map és reduce függvényt, ugyanis az adatbázisban megtalálható beépítve sok hasznos szűrési és aggregációs függvény. Viszont, az optimalizált keresések érdekében, természetesen futtatható a saját map és reduce is.

A 4.11 ábrán látható egy lekérdezés, amely a következőket hajtja végre:

- leszűri azokat a sorokat, amelyeknél létezik *dbTípus* kulcs és az értéke ráillik a *mongodb* reguláris kifejezésre, eltekintve a kis és nagybetűktől (SQL WHERE záradék)

- száz darabra csökkenti a sorok számát (SQL LIMIT záradék)
- az első húsz dokumentum kihagyásra kerül, és a huszonegyediktől választ ki száz darabot (SQL LIMIT vagy OFFSET záradék)
- a leszűrt dokumentumokat sorba rendezi, *létrehozva* mező alapján csökkenő sorrendben (SQL ORDER BY DESC záradék)

```
1 db.newCollection
2     .find({
3          "dbTípus": /mongodb/i
4      })
5     .limit( 100 )
6     .skip( 20 )
7     .sort({
8         "létrehozva": -1
9     })
```

4.11. ábra. Keresés MongoDB-ben

4.4.3. Fájltárolás

A MongoDB-ben egyik szolgáltatása a *GridFS*. A GridFS felelős a fájlok tárolásáért az adatbázisban. Ha MongoDB-be beszúrásra kerül egy fájl, akkor először az adatbázis szétdarabolja szeletekre (chunk), melyek dokumentumokként kerülnek eltárolásra, a szeletek maximális mérete 4 megabájt lehet.
A fájlok és a fájlok részei külön gyűjteményekbe kerülnek. A 4.12 ábrán látható egy eltárolt, közel 165 megabájtos zenefájl. A dokumentumban látható a fájl mérete, a neve, az MD5 hash értéke, a feltöltése ideje és az eltárolt szeletek mérete. A fájl szeleteinek mérete 262 148 bájt azaz az adott fájl 659 részletben van eltárolva.

```
{
    "_id": ObjectId("4d57b35ffc5448f447e47a21"),
    "filename": "Letöltések/zene.mp3",
    "length": 172677932,
    "chunkSize": 262148,
    "uploadDate": "Wed Apr 13 2011 11:33:12 GMT+0100 (CET)",
    "md5": "8cd867af63afe2fb1baa9693a9d3fbb2"
}
```

4.12. ábra. Egy MongoDB-ben eltárolt fájl

A 4.13 ábrán az első szelet dokumentuma látható, amely rendelkezik egy fájl azonosítóval ("files_id" mező, mely megmutatja melyik fájl része az aktuális dokumentum), egy számmal, hogy hányadik szelet ("n") és egy adat résszel ("data"), mely tartalmazza a bináris adatot.

```
{
    "_id": ObjectId("4d57b35f0d01e35a6e9fa302"),
    "files_id": ObjectId("4d57b35ffc5448f447e47a21"),
    "n": 0,
    "data": BinData type: 2 len: 262148
}
```

4.13. ábra. A 4.12 ábrán látható fájl egy része (chunk)

Miért ilyen a tárolás?

Ennek a tárolási módszernek az az előnye, hogy ha a webes alkalmazásban, ahol zenéket lehet hallgatni, a felhasználó a 3. perctől kívánja ezt megtenni, akkor nem kell az adatbázistól elkérni az egész fájlt, mindössze attól a szeletettől kell lekérdezni, amelyet valóban hallgatni akar a felhasználó.

Továbbá, ha az adatbázisban 30-40 gigabájtos fájlok tárolása történik, amelyeket a felhasználók le akarnak tölteni, akkor a relációs adatbázisban az egész fájlt le kell kérdezni, melyet az alkalmazás megpróbál betölteni a memóriába, viszont ha a szerver nem rendelkezik ennyi memóriával, akkor nem tudja visszaadni a felhasználónak.

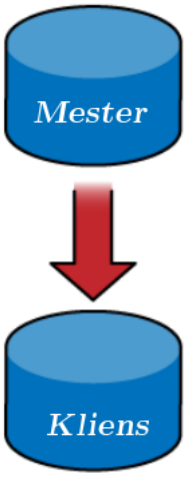
MongoDB-ben a 30-40 gigabájtos fájlnál sem fordulhat ilyen elő, mert ha az egész fájlt kéri el az alkalmazás, akkor is mindig csak egy szeletet ad át az adatbázis, azaz a memóriában maximum 4 megabájt lesz lefoglalva.

4.4.4. Replikáció

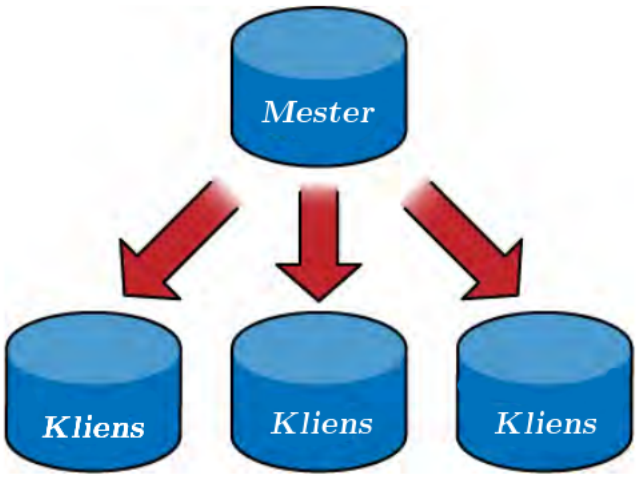
A MongoDB támogatja az aszinkron replikációt automatikus helyreállítással (hálózati vagy szerverhiba esetén). A replikációnak két fajtája van: mester-kliens, és a replica set alapú.

Mester-kliens

A mester-kliens általában egy mester adatbázisból és egy (4.14 ábra) vagy több (4.15 ábra) kliensből áll. Az egyetlen mester adatbázisba történik az adatok beszúrása, melyek automatikusan bekerülnek a kliens(ek) adatbázisába (a master egy olvasási eseményt küld a klienseknek). Amennyiben a kliensek kiesnek a replikációban (ez történhet az adatbázis szerver újraindulása miatt, hálózati forgalom hibáiból), a kliensek automatikusan újraszinkronizálják magukat.
Azt, hogy egy szerver mester vagy kliensként vesz részt az adattárolásban, a konfigurációban megadott érték dönti el.
A kliensek irányába történő szinkronizáció letiltható a mester szerverről, az adott kliens címének kizárásával.



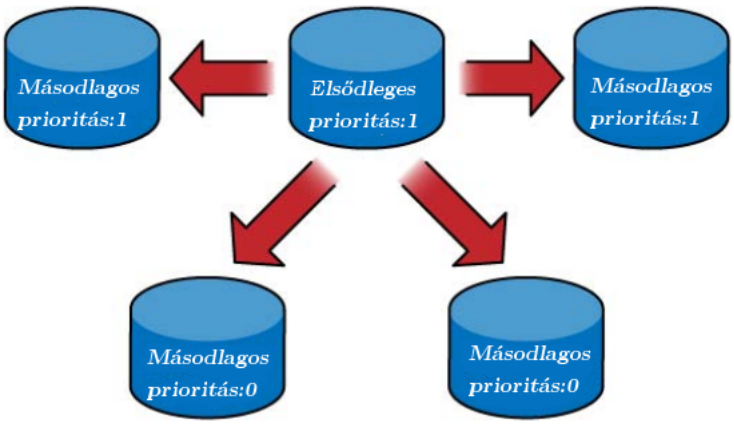
4.14. ábra. Egy mester, egy kliens replikáció a MongoDB-ben (Chodorow, 2011a)



4.15. ábra. Egy mester, három kliens replikáció a MongoDB-ben (Chodorow, 2011b)

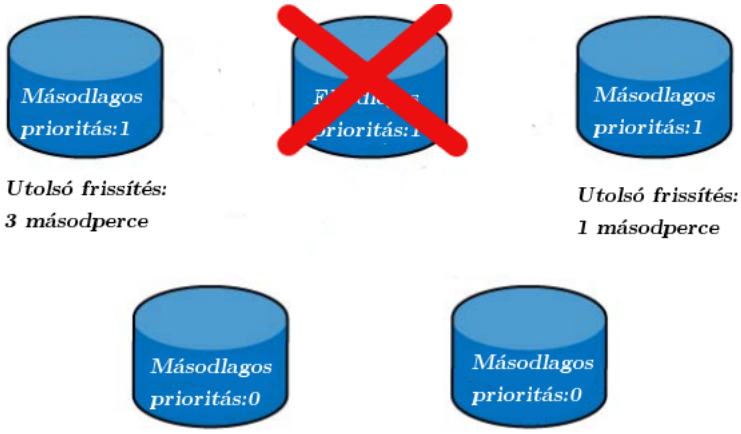
Replica set

A replica set tulajdonképpen egy mester-kliens replikáció automatikus helyreállítással. Ez azt jelenti, hogy a kommunikációban továbbra is egy mester (elsődleges) és egy kliens (másodlagos) szerver áll rendelkezésre, azonban hiba esetén a kliens mesterré válik, és a rekord beillesztések az új mester szerverbe folytatódnak. Amennyiben az eredeti mester helyreáll, az adatszinkronizáció elkezdődik, majd a végeztével minden visszaáll az eredeti állapotba, azaz az elsődleges szerver lesz ismét a mester, míg másodlagos klienssé válik.
A rendelkezésre állás tovább fokozható másodlagos szerverek priorizálásával (4.16 ábra).



4.16. ábra. Replica set egy lehetséges felépítése a MongoDB-ben (Chodorow, 2011c)

A szerverek ebben az esetben lehetnek akár klaszterek is, melynél az egész elsődleges klaszter kiesése után a legnagyobb prioritású másodlagos klaszter veszi át a mester szerepet, és így tovább egészen az utolsó működőképes szerverig vagy klaszterig.
Az elsődleges szerver kiesése után a MongoDB, a következő legmagasabb prioritású szervert választja, amennyiben egynél több ilyen van, akkor a legutolsó szinkronizáció óta eltelt idő alapján választ (4.17 ábra).



4.17. ábra. Döntéshozatal replica set replikációnál a MongoDB-ben, az elsődleges szerver hibája esetén (Chodorow, 2011d)

4.4.5. Tranzakciók

A MongoDB nem MVCC, hanem helyben-frissítés (update-in-place) alapú adatbázis-kezelő rendszer. A tranzakciók hiánya miatt a MongoDB használata elképzelhetetlen pénzügyi rendszerekben, viszont olyan alkalmazásoknál, ahol másodpercenként több száz vagy ezer rekordot kell létrehozni, lekérdezni, frissíteni vagy törölni a helyben-frissítés jobb választás lehet, ugyanis így kisebb az esélye a hibák felmerülésének a replikációkban. (Comparing MongoDB and CouchDB, 2011)

5. fejezet

Foursquare

5.1. Bevezetés

A foursquare egy helymegosztás alapú közösségi oldal, 2009 óta működik. A foursquare segítségével a városok gyorsabban felfedezhetőek, felhasználók által közkedvelt helyek hamarabb megtalálhatóak. Az oldal működésének elve, hogy ha a felhasználó elmegy egy étterembe, akkor be tud jelentkezni, hogy az adott helyen van, ilyenkor kaphat az étteremben kedvezményt illetve a barátai láthatják, hogy jelenleg hol tartózkodik. Mint a leírásból is kiderül az oldal alapvetően mobileszközökön használható jól, mert az aktuális földrajzi helyzethez képest ajánlja fel a rendszer helyeket. A helyzetmeghatározás történhet GPS pozíció, GSM tornyok beháromszögelése és a W3C HTML5 draftjában található geolokalizáció alapján is. (foursquare.com, 2011)
A foursquare-t azért választottam bemutatandó példának, mert dokumentum-orientált adatbázist használ, és az egyik legrégebbi MongoDB-t használó, komoly felhasználóbázissal rendelkező szolgáltatás. A közösségi oldal 2011. áprilisában mintegy 8 millió felhasználóval rendelkezett és ez a szám naponta körülbelül 35 ezerrel növekszik. A felhasználók naponta körülbelül 2,5 milliószor jelentkeznek be (az előző évben megközelítőleg fél milliárd bejelentkezés történt), ennek ellenére a szolgáltatás mögött álló foursquare Labs, Inc. mindössze 60 alkalmazottal rendelkezik. (foursquare.com, 2011b)
Az alkalmazás az Amazon EC2-ben[1] elhelyezett 25 darab szerveren fut, scalaban[2] írva. A rendszer indulásakor PostgreSQL adatbázist használt, azonban a felhasználók és bejelentkezések számának növekedésével a fejlesztők kénytelenek voltak az adatbázis réteg cseréje mellett dönteni, mivel a relációs rendszer skálázása (mind a szétosztás, mind a replikáció) komoly problémát jelentett. A választás a MongoDB-re esett, amelyhez a szerverek átalakítására volt szükség, ugyanis a NoSQL adatbázisok, a lemezre írás és lemezről olvasás helyett, a memóriát használják, ezzel csökkentve a lekérdezések idejét. Azonban a PostgreSQL a mai napig használatban van (az adatok rögzítése nem csak a MongoDB-be kerül írásra, hanem a relációs adatbázisba is), tehát egyfajta biztonsági mentésként üzemel.

[1] az induló alkalmazások (startup) egyik legkedveltebb felhőszolgáltatója
[2] Java és ML alapú programozási nyelv, Java virtuális gépen (JVM) fut

5.2. MongoDB a foursquare-nél

Ahogyan az előző pontban olvasható, a szervereket át kellett alakítani a MongoDB-hez, ez annyit jelent, hogy bővítették a memóriát 66 gigabájtig, amellyel elérték, hogy a legtöbb információ eléréséhez nem kell, a merevlemezről olvasni, ugyanis elfér a memóriában.

5.2.1. Előnyök

Skálázhatóság. A foursquare - mint jelenleg feltörekvő szolgáltatás - esetében a folyamatos felhasználónövekedés miatt, folyamatosan szükséges a szerverkapacitás bővítése. A hardveres bővítést az Amazon oldja meg, míg az új adatbázis-kezelő bekapcsolása a szolgáltatásba a foursquare dolga, amely szerencsére nem bonyolult MongoDB segítségével, mindössze be kell állítani az elosztást végző szerveren az új eszköz paramétereit, és onnantól kezdve az adatok szétosztása és lekérdezése automatikusan működik.

Geolokalizáció. Mivel a szolgáltatás alapja a lokalizált találat, és a MongoDB-ben megtalálható földrajzi koordináták indexelése, ezért a legközelebbi helyek megtalálásának és listázásának rövid válaszidejét nagyban elősegíti az adatbázisnak ez a funkciója. (Foursquare & MongoDB, 2010a)

Rövid válaszidők. A rövid válaszidőknek köszönhetően az alkalmazás több kérést tud indítani az adatbázis felé, amely a foursquare esetében 5000(!) kérés másodpercenként, mindezt 25 szerver segítségével oldják meg. (Q and A from Scaling foursquare with MongoDB, 2010)

5.2.2. Hátrányok

Indexelt mezők méretének limitálása. Az indexelt mezők korlátja egy kilobájt, a foursquare-nél ez komoly problémákat okoz, ugyanis az egy-egy helyhez több tíz vagy akár száz kulcsszó is tartozhat. Az indexelés limitálásával a kulcsszavak egy része lesz csak felindexelve, míg a többi kulcsszóra a keresés lehetetlenné válik. A foursquare a problémát a sok kulcsszóval rendelkező helyeknél, az indexelés kihagyásával kerülte meg. (Foursquare & MongoDB, 2010b)

6. fejezet

Összefoglalás

Az NoSQL adatbázisok kiváló megoldást nyújthatnak a mai szoftverek által támasztott igényekre, azonban fontos hangsúlyozni, hogy amíg a felhasználó nem ismeri a relációs rendszereket, illetve a relációs adatbázisok határait addig biztosan nincs szükség dokumentum-orientált adatbázisra.
Egy pénzügyi rendszerben, amely tranzakció-orientált és pesszimista zárolást alkalmaz, nincs szükség dokumentum-orientált adatbázisra, ráadásul (ha mégis bevezetnék) nem biztos, hogy az átállás megtérülne.
Viszont a közösségi oldalaknál, ahol a rugalmas adatkötések dominálnak, sok fájlt kell tárolni (fényképek, videók), illetve a válaszidők rendkívül fontosak, ott érdemes elgondolkodni a váltásban. A legnagyobb közösségi oldalak már felismerték ezeket a lehetőségeket és váltottak (foursquare), vagy épp folyamatban van váltás (Facebook, Twitter).

Felhasznált irodalom

Anderson J. Chris, Lehnardt Jan, Slater Noah (2010a). *CouchDB: The Definitive Guide*. O'Reilly Media, p. 39.

— (2010b). *CouchDB: The Definitive Guide*. O'Reilly Media, pp. 205–207.

Apache HTTP Server. Letöltve: 2011. április 22.
. URL: https://httpd.apache.org/.

CERN - CouchDB and Physics (Aug. 2010). Letöltve: 2011. április 22.
. URL: http://www.couchbase.com/sites/default/files/pdf/case-studies/CernCB-2010Aug.pdf.

Chodorow Kristina (2011a). *Scaling MongoDB*. O'Reilly Media, p. 127.

— (2011b). *Scaling MongoDB*. O'Reilly Media, p. 128.

— (2011c). *Scaling MongoDB*. O'Reilly Media, p. 135.

— (2011d). *Scaling MongoDB*. O'Reilly Media, p. 136.

Comparing MongoDB and CouchDB (Jan. 2011). Letöltve: 2011. április 10.
. URL: http://www.mongodb.org/display/DOCS/Comparing+Mongo+DB+and+Couch+DB.

CouchDB Replication (Oct. 2010). Letöltve: 2011. április 14.
. URL: http://wiki.apache.org/couchdb/Replication.

ECMAScript Language Specification (Dec. 2009). Letöltve: 2011. április 01.
. URL: http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf.

Facebook statiszika (Apr. 2011). Letöltve: 2011. április 24.
. URL: http://www.facebook.com/press/info.php?statistics.

Foursquare & MongoDB (May 2010a). Letöltve: 2011. április 15.
. URL: http://docs.google.com/present/view?id=dhkkqm6q_13gm6jq5fv.

Foursquare & MongoDB (May 2010b). Letöltve: 2011. március 22.
. URL: http://blip.tv/file/3704098.

foursquare.com (Apr. 2011a). Letöltve: 2011. április 20.

. URL: http://www.foursquare.com.

foursquare.com (Apr. 2011b). Letöltve: 2011. április 23.

. URL: https://foursquare.com/about.

Gajdos Sándor (2006). *Adatbázisok*. Műegyetemi Kiadó, p. 4.

Introduction to CouchDB Views (Jan. 2011). Letöltve: 2011. április 18.

. URL: http://wiki.apache.org/couchdb/Introduction_to_CouchDB_views.

Q and A from Scaling foursquare with MongoDB (Dec. 2010). Letöltve: 2011. március 25.

. URL: http://www.10gen.com/video/misc/foursquareqa.

Twitter Hits Nearly 200M Accounts, 110M Tweets Per Day, Focuses On Global Expansion (Jan. 2011). Letöltve: 2011. március 10.

. URL: http://blogs.forbes.com/oliverchiang/2011/01/19/twitter-hits-nearly-200m-users-110m-tweets-per-day-focuses-on-global-expansion/.

Who Leads the RDBMS Pack? Letöltve: 2011. március 28.

. URL: http://databases.about.com/library/weekly/aa060401a.htm.