Haaga-Helia
ammattikorkeakoulu Oy

Indexed Database ja Web SQL Database -rajapinnat

Samuel Kontiomaa

Opinnäytetyö
Tietojenkäsittelyn
koulutusohjelma
2016

Haaga-Helia
ammattikorkeakoulu Oy

Tiivistelmä

18.04.2016

Tekijä(t)
Samuel Kontiomaa

Koulutusohjelma
Tietojenkäsittely koulutusohjelma

Opinnäytetyön otsikko	**Sivu- ja liitesivumäärä**
Indexed Database ja Web SQL Database -rajapinnat	34 + 2

Opinnäytetyön otsikko englanniksi
Indexed Database and Web SQL Database APIs

Opinnäytetyössä tutustutaan palvelimien sijaan käyttäjäpäässä tapahtuvaan verkkosivujen tiedon varastoinnin ja käsittelyn tapoihin. Tutkimuksen kohteeksi valittiin JavaScript natiivit teknologiat; Indexed Database sekä Web SQL Database.

Teoriaosuudessa käydään läpi lyhyesti tietokantojen historiaa, kuinka fyysisestä mediasta digitaaliseen siirryttäessä tuli vastaan ongelmia tiedon hajanaisessa säilyttämisessä määrän kasvaessa jatkuvalla tahdilla. Tätä ongelmaa korjaamaan kehitettiin tietokannat. Teoriaosuus käsittelee relaatiotietokantoja jonka edustaja Web SQL Database on sekä NoSQL –tietokantoja, johon vuorostaan Indexed Database lukeutuu. Web SQL Databasea ja Indexed Databasea käsitellään tarkemmin omissa luvuissaan.

Tutkimusosuudessa kerrotaan rajapintojen käyttöönotosta, tutustutaan niiden toimintaan ja vertaillaan eroja samojen toiminnallisuuksien saamiseksi itse tehtyyn demo-sovellukseen. Lopuksi mitataan ja vertaillaan CRUD –operaatioiden viemää aikaa rajapintojen välillä. Tutkimuksessa ei käytetty haastatteluja eikä kyselyitä. Itse demoa ei esitellä yksityiskohtaisesti, vain rajapintaan tutustumisen puitteissa mutta koodi on vapaasti kaikkien ladattavissa ja käytettävissä.

Tutkimuksen tuloksista voidaan todeta, että Web SQL Database vastaa toiminnaltaan relaatiotietokantaa eikä todennäköisesti tuottaisi suuremmin hankaluuksia oppia näitä käyttäneille. Indexed Database vuorostaan vastaa toiminnaltaan eniten NoSQL:n dokumenttitietokantaa joihin mm. MongoDB lukeutuu.

Molemmista rajapinnoista käydään läpi esimerkki erilaisista CRUD tapahtumista ja perehdytään ongelmia aiheuttaneisiin tapahtumiin. CRUD –operaatioiden aikoja mitattaessa ilmeni, että Web SQL Database oli lähes poikkeuksetta Indexed Databasea nopeampi. Nopeusedusta huolimatta Web SQL Databasea on hankala suositella mihinkään pidemmän elinkaaren projektiin sen aktiivisen kehityksen lopettamisen johdosta.

Asiasanat
Indexed Database, Web SQL Database, rajapinnat, JavaScript

HAAGA-HELIA
University of Applied Sciences

Abstract

18.04.2016

Author(s)
Samuel Kontiomaa

Degree programme
Business Information Technology

Report/thesis title
Indexed Database and Web SQL Database APIs

Number of pages and appendix pages
34 + 2

The purpose of this thesis is to analyze information storage and -handling of website data on the client-side. The chosen technologies are JavaScript native APIs called Indexed Database & Web SQL Database.

The theoretical part of this thesis goes briefly over the history of databases, how problems arose with scattered information storage when moving from physical to digital format all the while quantity rose steadily. The databases were created to rectify this problem. The theory moves on to relational databases, in which the Web SQL Database belongs to and continues on to NoSQL databases where Indexed Database belongs in turn. The Web SQL Database and Indexed Database APIs are introduced more closely in their respective chapters.

The study describes the APIs deployment and functions, compares the differences between the APIs in order to get the same functionality while using a self-made application for demonstration purposes. Lastly the time taken by the APIs' CRUD -operations are measured and compared. The study didn't include interviews nor surveys. The detailed introduction of the demo application is out of scope, it is only used to showcase the use of the APIs, although the code for the application can be freely downloaded and used by anyone.

The study reveals that Web SQL Database is functionally equivalent to relational databases and shouldn't cause any major problems for developers who have used them before. Indexed Database, on the other hand, resembles document-oriented NoSQL databases the most, of which MongoDB is an example of.

Both APIs have an example of each CRUD operation while also focusing on API-specific problems that arose when developing the application. While measuring the CRUD-operation times, it was revealed that Web SQL Database was predominantly faster when compared to Indexed Database. Despite the speed advantage, it is inadvisable to recommend Web SQL Database to be used in any project with a long life cycle because it is no longer actively developed.

Keywords
Indexed Database, Web SQL Database, Application programming interface, JavaScript

Sisällys

Käsitteet

BSON (Binary JSON) on binääri-koodattu JSON dokumentti. JSON:in tavoin BSON tukee dokumenttien ja taulukoiden upottamista toisten dokumenttien ja taulukoiden sisään. (bsonspec.org.) BSON:ia käytetään mm. MongoDB:n tiedon tallennuksessa, koska se on kevyt, nopeasti käsiteltävissä ja sallii haut sekä indeksien luomisen koko dokumentin sisällölle. (mongodb.com.)

Eväste (Cookie) on pieni tekstitiedosto, minkä selain tallentaa käyttäjän tietokoneelle tämän käydessä tietyillä nettisivuilla. Evästeeseen tallennetaan tarpeiden mukaan tietynlaista tietoa käyttäjästä, kuten käyttäjän kirjautumisesta palveluun tai hänen ostoskoristaan nettikaupassa. Eväste lähetetään tai haetaan aina sivun lataamisen yhteydessä. Tämän lisäksi evästeiden maksimi koko on 4 kilotavua per nettisivu, mikä rajoittaa niiden käyttöä minkäänlaiseen tiedon hallintaan. (Zakas 2009.)

CRUD on akronyymi sanoista Create, Read, Update ja Delete. Sanoja käytetään puhuttaessa tietokannan eri operaatioista.

IndexedDB Indexed Database API on W3C:n ylläpitämä ja kehittämä standardi NoSQL tyyppisestä tietokannasta selaimelle. IndexedDB:tä suositeltiin standardiksi 8.1.2015. Web Storagen tavoin Indexed Database mahdollistaa suurien tietomäärien tallentamisen käyttäjän koneelle. Web Storagesta poiketen Indexed Database on tapahtumia (transaction) tukeva tietokanta, joka mahdollistaa mm. tietojen listaamista, muokkaamista ja poistamista hakuja käyttämällä. (Mehta ym 2015.)

JSON (JavaScript Object Notation) on kevyt JavaScriptiin perustuva kieliriippumaton tiedonvaihtokieli. JSON:ia on ihmisen helppo lukea sekä kirjoittaa ja tietokoneen helppo jäsentää ja luoda. JSON perustuu kahdenlaiseen rakenteeseen. Kokoelmaan avain-arvo pareja sekä arvoista koostuvaan listaan. (json.org.) JSON tukee kaikkia perus tietotyyppejä: numeroita, tekstiä, totuusarvomuuttujaa (boolean) sekä taulukkoja ja tiivisteitä (hash). (mongodb.com.)

NoSQL (non SQL/Not Only SQL) tietokannalla ei ole tarkkaa määritystä, pääsääntöisesti sillä tarkoitetaan uudehkoja tietokantoja, joissa ei ole suhteita taulujen välillä eivätkä käytä kyselykielenään SQL:ää. Tietokannat ovat useimmiten avointa lähdekoodia, niiden toimintaa voidaan jakaa clustereihin suuremman tehon saavuttamiseksi ja ne ovat kaaviottomia tai kaaviot ovat vähintään dynaamisia. (Fowler 2012.)

SQL Structured Query Language, rakenteellinen kyselykieli. SQL:n standardi määriteltiin ANSI:n (American National Standards Institute) toimesta vuonna 1986 ja adoptoitiin kansainväliseksi standardiksi vuonna 1987 ISO:n toimesta (International Organization for Standardization). (Connolly & Begg 2015, 191.)

SQL on suunniteltu toimiaan mahdollisimman pienellä vaivalla käyttäjälle, syntaksi on yksinkertainen ja helposti opittava. SQL toimii käyttäjän ja DBMS:n välissä, joko interaktiivisesti käsin kirjoitettuna tai sulautettuna sovellukseen, oli tietokanta mikä tahansa. SQL -kielellä kuvaat mitä haluat, DBMS:n tehtävä on selvittää miten ja mistä tieto löytyy. SQL:n toiminnat voidaan jakaa kuvaus- ja käsittelykieleen. (Connolly & Begg 2015, 192-193.)

Kuvauskieli (Data Definition Language) mahdollistaa relaatiotietokannassa relaatioiden, näkymien ja hakuindeksien luomisen, muuttamisen ja poistamisen (Laiho 1990, 14).

Käsittelykieli (Data Manipulation Language) mahdollistaa relaatiotietokannoissa relaation käsittelyä tietorivien lisäämisellä, hakemisella, päivittämisellä ja poistamisella, annettujen kriteerien mukaan (Laiho 1990, 31).

Web Storage mahdollistaa tiedon tallentamisen tietokoneelle selaimen käytettäväksi. Evästeisiin verrattuna, Web Storage pystyy tallentamaan enemmän tietoa eikä palvelimen tarvitse lähettää erillistä pyyntöä päästäkseen tietoihin käsiksi. Tiedon tallennus perustuu avain-arvo-pareihin. Arvot on mahdollista luokitella vain merkkijonoiksi (string), minkä takia mitään matemaattista laskentaa tiedolla ei voi suorittaa sellaisenaan. (W3Schools 1999.)

Web SQL Database on SQLiteen pohjautuva relaatiotietokanta toteutus selaimelle. W3C lopetti standardin kehittämisen 18.11.2010, koska SQLite oli ainoa käytetty tietokantatoteutus ja standardin kehittämiseksi vaadittiin useampia ratkaisuja (Hickson 2010).

XML (Extensible Markup Language) on yksinkertainen ja joustava tekstin tallennusmuoto, joka suunniteltiin alun perin vastaamaan laajamittaisen sähköisen julkaisun tuottamiin haasteisiin. (Quin 2015.)

1 Johdanto

Perinteisesti tietokannat sijaitsevat yrityksen omistamilla tai vuokraamilla palvelimilla konesaleissa. Tietokannat tallentavat esimerkiksi tietoa nettikaupan tuotteista ja kauppaa käyttävistä asiakkaista sekä heidän ostoksistaan ja ostoskoreistaan.

JavaScriptista ja Json:ista innostuneena minua kiinnosti lähteä selvittämään miten tiedon varastoiminen ja muokkaaminen onnistuu suoraan selaimessa ns. natiivisti, ilman erinäisiä kirjastoja tai ohjelmistokehyksiä (framework). Web Storagen kautta tutustuin Indexed Databaseen sekä Web SQL Databaseen, joita halusin tutkia tarkemmin.

Projektin alussa käsitellään Relaatio- ja NoSQL tietokantojen teoriaa perustetasolla; mitä ne ovat ja mihin niitä käytetään. Teorian jälkeen perehdytään itse Indexed Database sekä Web SQL Database rajapintoihin testaamalla niiden toimintaa CRUD -operaatioissa ottaen samalla ylös operaatioiden nopeuksia, testit toteutetaan käyttäen itse tehtyä demosovellusta. Molemmista rajapinnoista käydään niiden toiminnallisuudet läpi, tutustuen samalla itse demossa käytettyyn koodiin.

Projektin aikana ei käytetty haastatteluja tai kyselyitä. Tutkimusta varten luotu demo on vapaasti kaikkien ladattavissa ja käytettävissä GPLv3 -lisenssin ehtojen mukaisesti, mutta sitä käytetään projektissa ainoastaan työkaluna rajapintoihin tutustumista varten ja itse koodiin ei tutustuta sen tarkemmin muutamaa esimerkkiä lukuun ottamatta. (Kontiomaa 2016).

Projektin tuloksena syntyi teoriataustaa projektiin valituista kannoista, web-sovellus testejä ja vertailuja varten. Testien tulokset ja niistä johdetut vertailut sekä pohdinnat. Web SQL- ja Indexed Databaseen tutustumisen (sekä rajapintoja käyttävän demon luomisen) lisäksi tulin projektin aikana syventäneeksi osaamistani relaatiotietokannoista. Tämän lisäksi tutustuin erityyppisten NoSQL-tietokantojen toimintaan teoriatasolla sekä tulin testanneeksi MongoDB:tä, joka helpotti minua ymmärtämään Indexed Databasen toimintaa paremmin.

2 Tietokannan asema sovelluksessa

Tässä kappaleessa käydään läpi tietokannan käsitettä yleisellä tasolla sekä perehdytään relaatio- ja NoSQL tietokannan eroihin.

2.1 Historiaa

Tietokantoja edeltävänä tallennusmuotona voidaan pitää perinteisiä tiedostoja. Tiedostopohjaisissa tallennusmenetelmissä käyttäjiä palvelevat sovellukset määrittelivät ja hallitsivat itse oman tallennettavan tietonsa. Tiedostotallentamisen oli määrä korvata tiedon fyysinen arkistointi, koska tietoa käytettiin ja tallennettiin yhä enemmän. (Connolly & Begg 2015, 55-60.)

Siirrettäessä tietoa digitaaliseen muotoon, jaettiin tiedot eri yksikköjen välillä, sen sijaan, että kaikki tieto olisi kootusti tallennettu yhteen paikkaan. Tiedon hajauttaminen aiheutti sen, että samankaltaista tietoa tallennettiin useaan paikkaan. Tämä aiheuttaa turhaa toistoa sekä tarvittavan tiedon hakeminen useammasta lähteestä on hankalaa ja hidasta, puhumattakaan tilanteessa, jossa jotain tietoa pitäisi muuttaa tai päivittää useaan paikkaan. Edellä mainitut ongelmat kuluttavat työntekijöiden aikaa ja käytettäviä resursseja. Resurssiongelmien lisäksi teknisellä puolella ongelmia tuli, jos oli tarpeen muuttaa tallennettavan tiedon määrityksiä, kuten sallittujen merkkien määrää tai oli tarve lisätä täysin uutta tietoa jo olemassa oleviin tiedostoihin. Tiedon tallennusmuoto oli myös ongelma jos käytettävät sovellukset käsittelevät tallennettua tietoa eri tavalla, jolloin eri tietoja ei voinut välttämättä käsitellä samasta työpisteestä. Tiedostoista haettavaa tietoa ei myöskään pystynyt määrittelemään lennosta, vaan tiedosto piti lukea sellaisenaan tai ohjelmistokehittäjien piti itse ohjelmoida käytettäviä näkymiä, tämän lisäksi yhtä tiedostoa pystyi käsittelemään vain yksi henkilö kerrallaan. (Connolly & Begg 2015, 60-62.)

Tiedostopohjaisen tallennus teknologian puutteita korjaamaan nousi tietokanta sekä tietokannan hallintajärjestelmä. Tietokannassa tieto on keskitetysti kaikkien saatavilla samaan aikaan, tällä tavoin tietoa ei tarvitse jonottaa ja siihen pääsee käsiksi omalta työpisteeltä. Koska tieto on yhdessä paikkaa, sen päivittäminen onnistuu helposti eikä ylimääräistä toistoa ole. Tietokanta pitää myös kirjaa tallennetun tiedon tyypistä (teksti-, numero-, päivämäärä- tai jotain muuta tietoa), tästä johtuen tallennettu tieto on ohjelma riippumaton. (Connolly & Begg 2015, 62-63.)

Tietokannan hallintajärjestelmä (Database Management System - DBMS) on ohjelma, joka toimii käyttäjän sovelluksen ja tietokannan välissä. DBMS:n avulla käyttäjä voi luoda

tietokantoja, määritellä sen sisältöä ja rakennetta, lisätä, päivittää, poistaa ja hakea tietokannan tietoa. Edellä mainitut tietokannan manipuloinnit suoritetaan kyselykielillä, yleisin relaatiotietokannan käyttämä kyselykieli on SQL (Structured Query Language). (Connolly & Begg 2015, 64.)

Voisi ajatella, että tietokanta on kuin kirjasto. DBMS on kirjaston henkilökunta, joka pitää hyllyt siistinä ja kirjat järjestyksessä. SQL on varauksen tehnyt asiakas, joka tulee noutamaan varaustaan tai asiakas, joka tulee kirjastoon kysymään henkilökunnan apua kirjojen hakemisessa. Kirjat olisi mahdollista hakea itse, mutta se on paljon yksinkertaisempaa ja nopeampaa käyttää henkilökunnan apua.

Tietokanta on siis kokoelma pidemmälle aikavälille tallennettavaa tietoa, jota ylläpidetään ja hallitaan tietokantajärjestelmillä. Tietokantajärjestelmä antaa käyttäjälle mahdollisuuden luoda uusia tietokantoja, hakea tietoa olemassa olevista tietokannoista sekä muokata ja poistaa tallennettua tietoa. Tietokantajärjestelmän on pystyttävä tallentamaan suuria määriä tietoa ja tarjoamaan tallennettua tietoa usealle käyttäjälle samaan aikaan ilman riskiä tiedon eheydelle.

Vaikka Tietokanta + DBMS ratkaisivat monta ongelmaa tiedostopohjaiseen tallennukseen verrattuna, se ei ole täydellinen. Tietokannan rakenne on monimutkainen (complex), se täytyy suunnitella oikein käyttötarpeisiin sopiviksi tai vastaan tulee ongelmia ja aikaa vievä korjaustyö. Tietokanta voi viedä paljon tilaa ja konetehoja, tämän lisäksi kaupallinen käyttö ja ylläpito on maksullista. Koska tietokannassa tieto on varastoitu tietyllä tavalla, vanhan tiedon siirtäminen tietokantaympäristöön ei ole rahallisesti tai ajallisesti aina järkevää. Yksi tietokannan hyödyistä voidaan laskea myös heikkoudeksi; kaikki tieto on varastoitu keskitetysti yhteen paikkaan, kaikkien saataville. Jos jotain menee rikki, ei kukaan enää pääse tietoihin käsiksi. (Connolly & Begg 2015, 75-79.)

2.2 Relaatiotietokanta

Relaatiotietokanta ja relaatiotietokannan hallintajärjestelmä edustavat tietokantojen toista sukupolvea. Relaatiotietokanta pohjautuu Edgar Coddin kehittämään relaatiomalliin (relational data model). Relaatiotietokannassa kaikki tieto on loogisesti järjestetty relaatioihin (relation, SQL:ssä table, suomeksi taulu). Relaatio sisältää attribuutteja (attribute, SQL:ssä column, suomeksi sarake), jotka määrittelevät ja kuvaavat tallennettavaa tietoa. Monikko (tuple, SQL:ssä row, suomeksi rivi) sisältää tietoa, yksi tietoa sisältävä arvo per attribuutti. (Connolly & Begg 2015, 149.) Alla olevassa kuviossa havainnollistetaan relaation rakennetta (Kuvio 1).

```
mysql> select * from kirja;
+----------+---------------------------------------+----------------------------+--------------+-------------+-------+
| kirja_id | nimi                                  | genre                      | tekija       | kirjasto_id | vapaa |
+----------+---------------------------------------+----------------------------+--------------+-------------+-------+
|        1 | SQL                                   | ohjelmointikielet          | Martti Laiho |           1 |     0 |
|        2 | Getting started with Couchbase Server | tiedonhallintajärjestelmät | M. C. Brown  |           2 |     0 |
|        3 | MySQL : administrator's guide         | tietokantaohjelmat         | MySQB AB     |           2 |     0 |
|        4 | Linnunradan käsikirja liftareille     | tieteiskirjallisuus        | Douglas Adams |           1 |     0 |
+----------+---------------------------------------+----------------------------+--------------+-------------+-------+
4 rows in set (0.00 sec)
```

Kuvio 1. Esimerkki relaatiosta kuvitteellisen kirjaston tietokannassa.

Relaatiotietokannan relaatiot määritellään etukäteen kaavioilla (schema). Kaaviossa ilmoitetaan relaation ja attribuuttien nimet, attribuuttien sisältämän tiedon tyyppi. Relaatiotietokannassa jokaisella rivillä on oltava uniikki tieto, jolla tietoa voidaan hakea ja siihen viitata. (Ullman & Widom 1997, 85-87.) Seuraavassa kuviossa on kuvattu edellisen kuvion relaatio kaaviona (Kuvio 2).

```
mysql> describe kirja;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| kirja_id    | int(11)      | NO   | PRI | NULL    | auto_increment |
| nimi        | varchar(100) | NO   |     | NULL    |                |
| genre       | varchar(100) | NO   |     | NULL    |                |
| tekija      | varchar(100) | NO   |     | NULL    |                |
| kirjasto_id | int(11)      | NO   | MUL | NULL    |                |
| vapaa       | int(11)      | NO   |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

Kuvio 2. Kaavio yllä olevasta kirjarelaatiosta.

Relaatioiden välinen yhteys luodaan pää- ja viiteavaimella. Pääavain on uniikki tunniste relaation monikoille. Viiteavaimella viitataan toisen relaation pääavaimeen. Toisin sanoen viiteavaimilla toteutetaan relaatioiden väliset yhteydet. (Harkins, 2003.) Seuraavassa kuviossa on esimerkki, kuinka relaatioita luodaan (Kuvio 3).

```
CREATE TABLE kirjasto(
kirjasto_id INT NOT NULL AUTO_INCREMENT,
nimi varchar(100) NOT NULL,
osoite varchar(100) NOT NULL,
primary key (kirjasto_id)
);

CREATE TABLE kirja(
kirja_id INT NOT NULL AUTO_INCREMENT,
nimi varchar(100) NOT NULL,
genre varchar(100) NOT NULL,
tekija varchar(100) NOT NULL,
vapaa INT NOT NULL default=1,
kirjasto_id INT NOT NULL,
primary key (kirja_id),
foreign key (kirjasto_id) REFERENCES kirjasto(kirjasto_id)
);
```

Kuvio 3. Kirja -relaatio sekä tämän vaatiman kirjasto -relaation luontilauseet.

Relaatioon tallennetun tiedon on oltava mahdollisimman atomista, sitä ei pidä voida hienontaa yksinkertaisempaan muotoon, tämän lisäksi tietoa ei saa toistaa tarpeettomasti (Connolly & Begg 2015, 157). Tiedon laadullista ylläpitoa helpottamaan on suunniteltu normalisointi (normalization).

Normalisointi suoritetaan tietokannan suunnitteluvaiheessa, tutkimalla attribuuttien välisiä yhteyksiä. Tietokantaan tallennettava tieto käytetään testien läpi, jossa määritellään tietotarpeisiin nähden sopivat relaatiot. Normalisoinnilla pyritään minimoimaan attribuuttien määrä, liittämään yhteen sidoksissa olevia attribuutteja ja välttämään turhaa toistoa. Tämä helpottaa tietokannan käyttöä sekä ylläpitoa. Normalisointi on jaettu viiteen asteeseen, kolmanteen asteeseen normalisoiminen on suositeltavaa, neljäs ja viides aste ovat suunniteltu harvinaisempia tilanteita varten. (Connolly & Begg 2015, 452-453.)

Tietokanta tarjoaa mahdollisuuden useamman käyttäjän yhtäaikaisen tiedon käsittelyn. Päällekkäisten tapahtumien (transaction) välttämiseksi, on tietokannan hallintajärjestelmässä tapahtuman käsittelijä. Oikeanlaisen toiminnan takaamiseksi tapahtumien käsittelyssä, käytetään ACID -ominaisuutta. ACID koostuu sanoista Atomicity, Consistency, Isolation ja Durability. Atomisuudella varmistetaan, että tapahtuma onnistuu kokonaisuudessaan tai ei laisinkaan, mikään toiminta ei jää puolitiehen. Eheydellä taataan, että tietokanta pysyy eheänä tapahtumasta toiseen, tapahtuman on noudatettava tietokannalle annettua määrityksiä. Eristävyydellä varmistetaan, että päällekkäisten tapahtumien vaikutukset eivät eroa siitä jos ne tapahtuisivat peräkkäin. Pysyvyys takaa, että tapahtuman suoriutuessa onnistuneesti, se ei katoa edes vian ilmetessä. (Ullman & Widom 1997, 12.)

2.3 NoSQL tietokanta

Tiedon tallennuksessa relaatiotietokanta on ollut suosituimpia ratkaisuja 80-luvulta lähtien. Ongelmat tiedon mallintamisessa ja tietokannan skaalautuvuuden rajoitteet palvelinten välillä suurien tietomäärien alla on johtanut uusien NOSQL –nimeä kantavien tietokanta-projektien syntymiseen. NOSQL (Not Only SQL) on hyvin laaja käsite joukosta tallennus-ratkaisuja, jotka eivät noudata relaatiomallia eivätkä käytä kyselykielenään SQL:ää. NoSQL –tietokannat voidaan jakaa neljään kategoriaan tietomallin/tiedon tallennusratkai-sun perusteella (datamodel): avain-arvo-, dokumentti-, sarake- ja graph tietokantoihin. Relaatiotietokannan ACID-ominaisuutta vastaa NoSQL tietokannoissa BASE-ominaisuus (Basically, Available, Soft-state, Eventually consistent). (Neubauer, 2010.)

ACID:in tarjotessa tiedon yhdenmukaisuutta, keskittyy BASE saatavuuden tarjoamiseen. BASE –järjestelmien ensimmäinen prioriteetti on sallia uuden tiedon tallentaminen, hetkel-lisen ristiriitaisuudenkin uhalla (out of sync). BASE järjestelmillä on taipumus olla yksinker-taisempia ja nopeampia, koska niiden ei tarvitse keskittyä resurssien lukitsemiseen ja avaamiseen. Niiden tehtävä on pitää prosessit toiminnassa ja käsitellä virheet myöhem-min. Tietokantakirjoituksia ei lukita, minkä takia luettava tieto ei ole hetki hetkeltä ajan tasalla. (McCreary & Kelly 2013, 27-28.)

Tilanteissa, joissa tallennetun tiedon määrä ylittää palvelimien kapasiteetin, voidaan tieto-kanta jakaa paloihin, jotka sitten jaetaan uusille palvelimille. Tapahtumaa kutsutaan sirpa-loimiseksi (sharding), tämä toimenpide on tärkeä NoSQL tietokannan toiminnalle ja on automatisoitavissa. Tietokantojen tiedon sirpalointi voidaan tehdä monella eri kriteerillä, esimerkiksi tarvittavan tiedon, maantieteellisen sijainnin mukaan tai täysin satunnaisesti. (McCreary & Kelly 2013, 28-30.)

Replikoinnissa tieto jaetaan eri solmujen (node) välillä, nopeuttamaan tiedon lukemista ja kirjoittamista. Isäntä-Orja asetelmassa isäntä vastaa tiedon tallennus tapahtumista ja orjat tiedon lukemisesta. Kun tietokannan sisältöön on tullut muutos, kopioi isäntä päivitykset orjille. Asetelma on paras, kun tietokannalta vaaditaan suuria määriä luku-operaatioita. Lisäetuna tietokannan tieto on hyvin varmuuskopioitu. Isäntä –solmun rikkoutuessa mene-tetään vain mahdollisuus tehdä uusia päivityksiä ja ne päivitykset, joita isäntä ei ehtinyt kopioida orjille. Isäntä –solmun valinta voidaan automatisoida solmuryppäälle, jolloin kirjoi-tus-toiminta saadaan palautettua nopeasti. Vertais-asetelmassa kaikki solmut ovat sa-manarvoisia. Jokainen solmu vastaa luku- ja kirjoitus –operaatioista tietokantaan. Asetel-ma nopeuttaa tiedon luku- ja kirjoitusnopeutta, mutta vaaraksi nousee tiedon yhtenäisyy-

den hetkellinen rikkoutuminen. Kun tietoa voidaan tallentaa useassa paikassa samaan aikaan, voi tiedon tallennukselle tulla päällekkäisyyksiä. Yksi tapa estää päällekkäisyydet on, solmujen välinen koordinointi kirjoitustapahtumissa. Riittää, että enemmistö solmuista on yhtä mieltä tiedon tallentamasta solmusta, hintana on solmujen välinen ”ylimääräinen” verkkoliikenne. (Sadalage & Fowler 2012, 40-43.)

Sirpalointi ja replikointi on mahdollista toteuttaa yhtä aikaa. Isäntä-Orja replikoinnissa ja sirpaloinnissa tieto on jaettu osiin, jokaisella osalla on oma isäntänsä. Asetuksista riippuen yhden osion isäntä voi toimia toisen osion orjana. Vertais replikoinnissa ja sirpaloinnissa on hyvä käyttää kolminkertaista replikointia. Tämä tarkoittaa, että jokainen sirpale on olemassa kolmessa eri solmussa. Yhden solmun kaatuessa, kopioidaan ”menetetty” sirpale toiseen solmuun, jotta kopio kerroin pysyy samana. (Sadalage & Fowler 2012, 43.)

2.3.1 Avain-Arvo -pohjainen tietokanta

Avain-Arvo tietokannat ovat yksinkertaisin NoSQL nimeä kantavan tietokannan kategoria. Tietokannasta voidaan hakea arvoa tunnisteen (key) perusteella, tallentaa tunnisteille arvoja ja poistaa näitä pareja. Jos ajatellaan avain-arvo kantaa relaatiokantana; sisältäisi relaatio vain id:n ja arvon. Arvoon voi tallentaa tietoa vain merkkijonona. Tallentaessa luodaan uusi rivi id:lle ja arvolle. Jos tunniste on jo olemassa, korvataan sitä vastaava arvo uudella. Avain-arvo parit tallennetaan binääri muodossa (blob) tietokantaan ilman sisällön analysoimista, sovelluksen vastuulle jää sisällön ymmärtäminen ja käsittely. Koska kaikki tallennettu tieto kuuluu pääavaimelle, suorituskyky on pääsääntöisesti loistava ja helposti skaalattavissa. (Sadalage & Fowler 2012, 81-88.)

2.3.2 Dokumenttipohjainen tietokanta

Dokumenttitietokannoissa tieto haetaan ja varastoidaan nimensä mukaisesti dokumenteista, dokumentit voivat olla mm. XML, JSON tai BSON dokumentteja. Tallennettujen dokumenttien rakenteiden ei tarvitse olla täsmälleen identtisiä, samankaltaisuus riittää. Kaaviota ei ole, joten tietokannan eri dokumentit voivat sisältää eri tietoa toisiinsa nähden. Toisin kuin relaatiotietokannoissa, dokumentit eivät voi sisältää tyhjiä arvoja ja kaaviottomuuden ansiosta uusia tietoja voidaan lisätä ilman määrittelymuutoksien vaatimaa lisätyötä. Avain-arvo kannan tapaan, tietokannasta löytyy uniikkeja avaimia, joiden arvoina nämä dokumentit toimivat. Dokumenttitietokannoista ei löydy relaatiotietokannan kaltaista tapahtumien (transaction) käsittelyä. Relaatiotietokannassa voidaan varmistaa koko tapahtuman onnistuminen, joka voi sisältää useamman taulun. Dokumenttitietokannoissa on ns. atominen tapahtuman käsittely, jolla voidaan taata yhden dokumentin sisäisen tapahtuman onnistuminen, ei kokonaisuutta. (Sadalage & Fowler 2012, 89-93.)

2.3.3 Sarakepohjainen tietokanta

Sarakepohjaisessa tietokannassa tieto tallennetaan sarakeryhmiin (column family) riveinä, joissa on useampi sarake yhdistettynä riviavaimeen. Kokoelma yhdessä käsiteltäviä rivejä muodostavat sarakeryhmän. Sarakeryhmät muodostuvat samankaltaisesta tiedosta, jota tavallisesti halutaan käsitellä samaan aikaan. Sarakeryhmät voidaan eritellä tavallisiin- ja supersarakeryhmiin. Tavalliset sarakeryhmät ovat relaatiotietokannan relaation kaltaisia; relaation monikolla on uniikki id sekä attribuutteja vastaavat arvot. Supersarakeryhmissä sarakkeilla voi olla omia sarakkeita. Relaatiotietokannassa relaatiolla voisi olla attribuutit: etunimi ja sukunimi, supersarakeryhmässä voisi olla sarake nimi, jolla on alasarakkeina etunimi ja sukunimi. Koko sarakeryhmän tieto haetaan aina kokonaisuudessaan tarpeista huolimatta. Dokumenttitietokannan tavoin tapahtumat ovat atomisia, se koskee vain yhtä riviä kerrallaan. (Sadalage & Fowler 2012, 99-104.)

2.3.4 Graph -tietokanta

Graph tietokannat ovat suunniteltu tallentamaan ja ylläpitämään tallennetun tiedon välisiä yhteyksiä. Kokonaisuuksien väliset yhteydet tallennetaan sellaisenaan, relaatiokannassa vastaavan toiminnallisuuden saamiseksi tarvittaisiin oma relaatio, johon kerättäisiin näitä kokonaisuuksien välisiä suhteita. (nosqlguide 2014b.) Teoriassa graph rakenne on normalisoitavissa relaatiotietokannan käyttöön, voi se kuitenkin vaikuttaa negatiivisesti tiedon hakunopeuteen rekursiivisissa rakenteissa. Jokaisessa haussa pitäisi turvautua relaatioiden välisen tiedon hakemiseen ja yhdistämiseen, mikä on tietokannalle raskasta. Vaikka graph tietokantaa voidaan käyttää samoissa tilanteissa, kuin relaatiotietokantaa, ei se ole tätä korvaamassa. Sopivaa tietokantaa valittaessa, on tärkeä määrittää tallennettava tieto ja kuinka sitä tullaan käyttämään. (Neubauer 2010.)

Kokonaisuuksia Graph tietokannassa kutsutaan solmuiksi (node). Solmut organisoidaan toistensa välisten suhteiden perusteella, mikä mahdollistaa muodostuneiden kaavojen (pattern) tutkimisen. Solmut muodostuvat ominaisuuksista, kuten nimi. Solmujen välisien suhteiden tyypin kuvaamisen lisäksi voidaan myös kuvata suhteen suuntaa, kuten yhdensuuntainen esineen omistaminen tai kaksisuuntainen ystävyys henkilöiden välillä. Kyselyiden tuottaminen graph tietokannassa on hyvin vapaata, hakuja suoritetaan kyselemällä yhtä tai useampaa eri solmujen välistä suhdetta. Suhteita solmujen välillä voidaan lisätä ja poistaa vapaasti ilman muutoksia itse tietokannan rakenteeseen. Sen lisäksi, että haetaan tietoa eri solmujen välisistä suhteista, voidaan vertailla yhtäläisyyksiä kahden eri solmujen suhteissa muihin solmuihin. Solmujen välisiin suhteisiin voidaan solmujen tavoin lisätä ominaisuuksia, kuten työntekijän rooli, ystävien tutustumisen päivämäärä tai lainatut tavarat. Tietyt Graph tietokannat, kuten Neo4J tukevat ACID:ia ja vaativat tässä tapauksessa

kirjoituksille tapahtuman. Haut voidaan suorittaa ilman tapahtumaa. (Sadalage & Fowler 2012, 111-115.)

2.4 Cap teoreema

Cap teoreema on Eric Brewerin esittämä ja Seth Gilbertin sekä Nancy Lynchin tukema teoria kolmesta tietokannan ominaisuudesta: yhdenmukaisuus (consistency), saatavuus (availability) sekä osion kestävyys (partition tolerance). (Sadalage & Fowler 2012, 53.) Yhdenmukaisuudella tarkoitetaan tässä tapauksessa sitä, että asiakkaat saavat yhdenmukaisia hakutuloksia vaikka tieto olisi replikoitu usealle osiolle. Saatavuudella tarkoitetaan päivitysten mahdollistamista kaikissa tilanteissa ilman viiveitä, vaikka joku osioista olisikin alhaalla. Osion kestävyydellä tarkoitetaan järjestelmän kykyä vastata asiakkaiden kyselyihin, vaikka kommunikaatio tietokannan osioiden välillä katkeaisi. (McCreary & Kelly 2013, 30-31.)

Teoreeman mukaan kolmesta edellä mainitusta ominaisuudesta, voi käytössä olla kerrallaan kaksi. Yhden palvelimen kokoonpano on esimerkki saatavuuden ja yhdenmukaisuuden yhdistelmästä, suurin osa relaatiotietokannoista toimii tällä periaatteella. Valinta haluttujen ominaisuuksien välillä ei aina ole täysin kiveen hakattu, valinnoissa voidaan tehdä kompromisseja. Yhdenmukaisuuden ja saatavuuden välillä esimerkiksi voidaan määrittää isäntä-orja rakenteita palvelimien välille. Yhteyden katketessa palvelinten välillä, isäntä – palvelimella on uusin tieto ja oikeus tallentaa muutoksia tarpeen mukaan, orja – palvelimella on yhteyden katkeamista edeltävä tieto käyttäjien luettavana, mutta ei mahdollisuutta uuden tiedon tallentamiselle ennen, kuin yhteys on korjattu. Tässä tapauksessa tieto on saatavissa vikatilanteissa, mutta saatu tieto ei välttämättä ole ajan tasalla. Jos kompromisseja ei tehdä tilanne olisi joko: a) yhteyden katketessa tieto on vain luettavissa ja kukaan ei saa tehdä muutoksia (yhdenmukaisuus) tai b) molemmat palvelimet saavat tallentaa uutta tietoa, jolloin tulee päällekkäisiä kirjoituksia (saatavuus). Kompromisseja voidaan myös tehdä tietyissä tilanteissa kestävyyden kanssa. Esimerkiksi nettipalvelun sessiota ja sen muutoksia ei välttämättä haluta tallentaa joka muutoksen jälkeen, varsinkin jos käyttäjiä on paljon. Palvelun käyttö on tällöin nopeampaa, mutta osa muutoksista ei välttämättä vikatilanteissa tallennu. (Sadalage & Fowler 2012, 53-57.)

3 Web-tietokantastandardit

Tässä kappaleessa käydään läpi Web SQL Database ja Indexed Database teknologiat. Kappaleessa tehdään myös pikainen katsaus World Wide Web Consortiumiin, joka yhdessä yhteistyökumppaneidensa kanssa luo ja ylläpitää eri verkkostandardeja.

World Wide Web Consortium on kansainvälinen yhteisö, jossa monen eri tahon yhteistyön toimesta kehitetään verkon standardeja (The World Wide Web Consortium 1990a). W3C kehittää yhteistyökumppaneidensa kanssa teknisiä määrityksiä ja suosituksia HTML5:n ohella mm. CSS:lle, SVG:lle, WOFF:lle ja XML:lle (The World Wide Web Consortium 1990b). W3C tarjoaa myös HTML –dokumenttien sekä CSS –tyylitiedostojen validointipalvelua, josta voi selvittää koodissa esiintyviä virheitä.

W3C:n dokumentaatiosta paljastuu, että Web Storage, Web SQL Database ja Indexed Databasen kehitys alkoi jokseenkin samanaikaisesti. Ensimmäisenä Web Storage huhtikuussa 2009 Googlen toimesta. (Hickson 2009a.) Syyskuussa samana vuonna omaksi projektikseen irtautui Web Database (Hickson 2009b), nykyinen Web SQL Database (Hickson 2010). Syyskuun lopussa alkoi Oraclen toimesta WebSimpleDB API:n kehittäminen (Mehta 2009), nykyisin tunnettu Indexed Database (Mehta ym 2015). Web SQL Database on selaimessa toimiva tietokanta, joka käyttää SQL:n variaatiota ja Indexed Database vuorostaan NoSQL:n toimintaa muistuttava olio varasto.

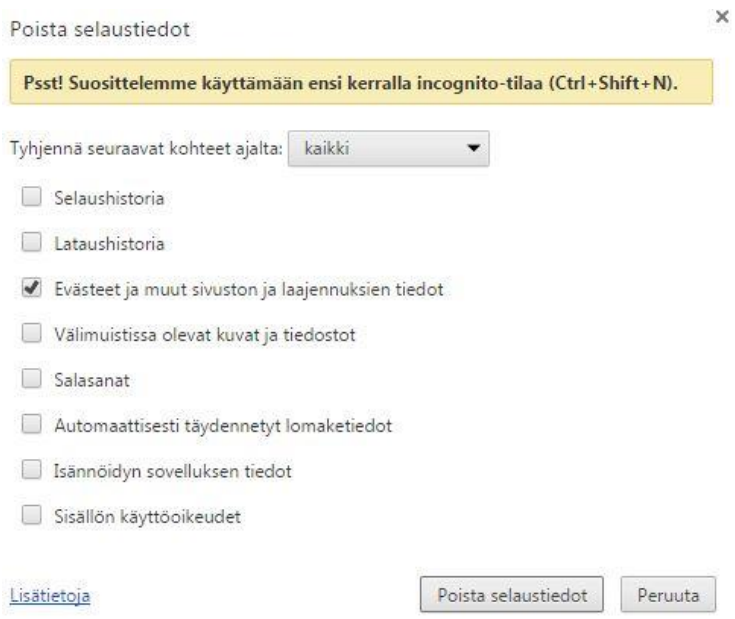
3.1 Web Storage

Web Storage on evästeiden tavoin tapa tallentaa avain-arvo-pareja selaimen käyttöön, käyttäjän koneelle. Web storagella on kaksi eri tallennusmenetelmää; session- ja localStorage. SessionStorage toimii vain kyseisessä selaimen välilehdessä, jolloin voidaan mm. tehokkaasti välttää ostoksien tai varauksien tekeminen kahteen kertaan jos nettisivu on avattu useammalle välilehdelle. Jokaisella avatulla sivulla on oma istuntonsa, evästeet sen sijaan ovat saman sivun käytössä useammallakin selaimen välilehdellä. LocalStorage toimii samalla sivulla useammallakin välilehdellä eikä ole istunto riippuvainen, eli tieto ei katoa vaikka selain suljettaisiin. Tämä mahdollistaa evästeisiin verrattuna suuremman tallennettavan tiedon määrän. Avaimet ja arvot koostuvat merkkijonoista, myös tyhjiä merkkijonoja voi käyttää. Web Storage on W3C:n suositusehdokas tulevaksi standardiksi 9.7.2015 lähtien. (Hickson 2015.)

3.2 Web SQL Database API

Web SQL Database API mahdollistaa SQL:ää tukevan tietokannan käytön käyttäjän selaimessa. Kaikki CRUD –tapahtumat ovat tuettuina. Jokaisella alkuperällä (domainilla) on omat tietokantansa. Jokaisella tietokannalla on nimi ja versionumero. Tietokantoja ei voi listata tai poistaa. Tietokantojen versionumero mahdollistaa kaavion muuttamisen rikkomatta sivuston toimintaa. Tietokannan nimi koostuu merkkijonosta, kirjainkoot otetaan huomioon, myös tyhjä merkkijono on sallittu tietokannan nimi. Web SQL:n tapahtumat voidaan suorittaa synkronisesti tai asynkronisesti, molemmissa vaihtoehdoissa on tuki tapahtuman hyväksymiselle (commit) ja peruutukselle (rollback). Commit ja Rollback –operaatiot ovat sisäänrakennettuja ominaisuuksia eikä niitä voi itse hallinnoida. Jos tapahtuman kaikki kyselyt onnistuvat, kutsutaan commit, epäonnistumisen sattuessa kaikki muutokset peruutetaan. Web SQL Database käyttää taustalla (backend) Sqliten versiota 3.6.19. Rajapinta ei ole tällä hetkellä aktiivisen kehityksen alla eikä W3C:n suosittelema standardi. Standardin määrittely keskeytettiin, koska taustalle vaadittiin Sqliten lisäksi muita toteutuksia (Hickson 2010.)

Tietokanta voidaan poistaa kokonaan tyhjentämällä selaushistoria ja valitsemalla ”Evästeet ja muut sivuston ja laajennuksien tiedot” kuten alla olevassa kuviossa on nähtävissä (Kuvio 4).



Kuvio 4. Web SQL Databasen sekä Indexed Databasen poistaminen tyhjentämällä selaushistoria.

Tietokannan poistaminen on mahdollista toteuttaa myös selaimen kansiosta käsin. Windows Vistasta lähtien Chrome tallentaa tietokannat domainkohtaisesti

”C:\Users\[käyttäjä]\AppData\Local\Google\Chrome\User Data\Default\databases” –kansioon.

Vaikka tietokannalla on versionumero ja täten mahdollisuus päivittää schemaa, on se tehty erittäin hankalaksi. Blogissaan Max Aller (Aller 2010.) on toteuttanut päivityksen avaamalla ensin tietokannan tyhjällä versiolla (tietokannan avaaminen epäonnistuu jos käyttäjän versio on pienempi, kuin nykyinen eikä päivitystä näin ollen voida suorittaa), jonka jälkeen tarkistetaan käyttäjän tietokannan versio ja aloitetaan päivitys. Tietokanta päivitetään versio kerrallaan, kunnes uusin on käytössä.

3.3 Indexed Database API

Indexed Database on tietokanta, joka ylläpitää yksinkertaisia avain-arvo pareja sekä olioita. Tämän lisäksi tallennettua tietoa on mahdollista indeksoida, jonka ansiosta haluttuja arvoja voidaan hakea ilman avainta. Hakujen suorittamiseksi, on indeksien oltava jo luotuna etukäteen. Tietokannat ovat alkuperä riippuvaisia. Tallennettuun tietoon pääsee käsiksi vain oltaessa samassa sivustossa, jossa tieto on alun perin tallennettu. Jokaiselle tietokannalle annetaan nimi ja versionumero. Tietokannan versioinnilla mahdollistetaan muutoksien ja päivityksien tekeminen. (Mehta ym 2015.)

Indexed Database käyttää ensisijaisena tallennusratkaisunaan oliovarastoja (verrattavissa relaatiotietokannan tauluihin). Tiedot tallennetaan tietueisiin (Avain-Arvo pari, joka voitaisiin mieltää relaatiotietokannassa sarakkeen Attribuutti-Arvo parina), jotka ovat osana listaa (vastaa relaatiotietokannan riviä). Tapahtumaa käytetään tietokannassa säilytettävän tiedon käsittelyyn. Se syntyy, kun selain ottaa yhteyttä tietokantaan. Tapahtumalle annetaan tyyppi, joka voi olla luku, luku ja kirjoitus tai tietokannan päivitys, eli versiomuutos. Tämän lisäksi tapahtumalla on tietty näkyvyysalue, jolla määritetään mihin tietoihin sillä on pääsy. Osoitin (cursor) on mekanismi, jota käytetään tietueiden iteroimiseen haettaessa tietoa useammasta listasta. Indexed Database on W3C:n suositus ja näin ollen lasketaan verkon standardiksi. (Mehta ym 2015.)

Web SQL Databasen tavoin, Indexed Databasen poistaminen on mahdollista toteuttaa tyhjentämällä selaushistoria sekä selaimen kansiosta käsin. Windows Vistasta lähtien Chrome tallentaa tietokannat domainkohtaisesti ”C:\Users\[käyttäjä]\AppData\Local\Google\Chrome\User Data\Default\IndexedDB” –kansioon.

4 Tutkimus

Aloitan tutkimuksen käymällä lyhyesti läpi Web SQL Database sekä Indexed Database rajapintojen käyttöönottoa. Vertailen rajapintoja keskenään tilanteissa, joissa ohjelmoinnin puolella oli tarvetta tehdä huomattavia muutoksia saman toiminnallisuuden aikaansaamiseksi. Viittaan myös kokemuksiani perinteisempiin tietokantoihin tilanteissa, joissa rajapintojen toiminnallisuus ei täysin taivu näiden tasolle.

Tutkimus jatkuu testaamalla molemmista rajapinnoista tietokannan perustoimintoja; Tiedon lisäämistä, -hakemista, -muokkaamista sekä -poistamista. Testi toteutetaan itse tehdyllä kirjastodemo-sovelluksella, tulen myös mittaamaan CRUD –operaatioiden viemää aikaa molemmista rajapinnoista ja vertaamaan niitä toisiinsa (Kontiomaa 2016).

Demo luotiin HTML:ää, CSS:ää sekä JavaScriptiä käyttäen eikä sisällä mitään kirjastoja tai ohjelmistokehyksiä. Demo toimii suoraan ilman mitään erityisempiä asennuksia, avaamalla index.html –tiedosto haluttua rajapintaa tukevalla selaimella, kuten Google Chromella tai Mozilla Firefoxilla (Firefox tukee vain Indexed Databasea). Indexed Database on oletuksena valittu rajapinta, joka täytyy vaihtaa käsin kaikkiin html tiedostoihin jos haluaa käyttää Web SQL Databasea tämän sijasta.

Tiedon tallennus sovelluksessa alkaa sivukohtaisen JavaScript funktion kutsumisella. Riippuen millä sovelluksen sivulla ollaan, suoritetaan tarvittava käsittely, kuten lomakkeen tietojen kerääminen. Kun kaikki tarvittava tieto on valmiina, avataan yhteys tietokantaan ja suoritetaan uusi tietokantatapahtuma.

Nopeuksien mittaaminen demossa tapahtuu vähentämällä tapahtuman onnistumiseen kuluneesta ajasta tapahtuman alkamisaika (Tapahtuma onnistui - Tapahtuma alkoi) käyttämällä JavaScriptin ”performance.now()” -komentoa.

Create testissä tullaan tallentamaan 100 000 käyttäjää Json –tiedostosta tietokantoihin. Tallennettava tieto koostuu kirjaimista ja numeroista, jotka on generoitu etukäteen ja tallennettu Json –tiedostoon tätä testiä varten. Create testin kanssa tulee Chromessa pieni ongelma, koska tiedostojen ajaminen ei ole paikallisesti sallittua (edes käyttämällä ”—allow-file-access-from-files” –lippua). Tästä johtuen käytin sovellusta palvelimelta käsin kaikkien testien ajan. Seuraavassa kuviossa esitellään pieni osa yllä mainitusta Json -tiedostosta (Kuvio 5).

```
{
    "users":[
{"firstname":"a1","lastname":"a1","address":"a1","email":"a1","card":"a1"},
{"firstname":"b1","lastname":"b1","address":"b1","email":"b1","card":"b1"},
{"firstname":"c1","lastname":"c1","address":"c1","email":"c1","card":"c1"},
{"firstname":"d1","lastname":"d1","address":"d1","email":"d1","card":"d1"},
{"firstname":"e1","lastname":"e1","address":"e1","email":"e1","card":"e1"},
{"firstname":"f1","lastname":"f1","address":"f1","email":"f1","card":"f1"},
{"firstname":"g1","lastname":"g1","address":"g1","email":"g1","card":"g1"},
{"firstname":"h1","lastname":"h1","address":"h1","email":"h1","card":"h1"},
{"firstname":"i1","lastname":"i1","address":"i1","email":"i1","card":"i1"},
{"firstname":"j1","lastname":"j1","address":"j1","email":"j1","card":"j1"},
{"firstname":"k1","lastname":"k1","address":"k1","email":"k1","card":"k1"},
{"firstname":"l1","lastname":"l1","address":"l1","email":"l1","card":"l1"},
```


Kuvio 5. Ensimmäiset rivit Json –tiedostosta.

Read testissä haetaan kaikki luodut käyttäjät tietokannasta. Update testi lisää luoduille käyttäjille muokattu -merkinnän (joka näkyy käyttäjän profiilissa). Delete testi poistaa tietokantojen kaikki käyttäjät. Molempien rajapintojen operaatioista mitattiin hitain sekä nopein aika.

Selain ylläpitää tietokantoja käyttäjän tietokoneen kovalevyllä. Tietokantoihin pääsee käsiksi alkuperäkohtaisesti (domain). Tämä tarkoittaa sitä, että tietokantoja voidaan käsitellä vain sen domainin alla, jossa tietokanta on luotu, olkoon se internetsivustolla tai itse käyttäjän koneella. Web SQL Database säilyttää tallennettua tietoa Windowsissa (Vistasta lähtien) Chrome-selaimella seuraavassa sijainnissa: ”C:\Users\[käyttäjä]\AppData\Local\Google\Chrome\User Data\Default\databases”. Indexed Databasessa vastaava sijainti on: ”C:\Users\[käyttäjä]\AppData\Local\Google\Chrome\User Data\Default\IndexedDB”. Alla olevasta kuviosta (Kuvio 6) näkyy Indexed Databasen tapa jäsennellä tallennettua tietoa domainkohtaisesti. Ensimmäiseen kansioon tallennettu tieto on tallennettu käyttämällä sovellusta tietokoneella paikallisesti.

Kuvio 6. Indexed Databasen tallentamat tiedot domainkohtaisesti.

Testit toteutettiin Googlen Chrome –selaimella, koska se tukee molempia rajapintoja. Chromessa on mahdollista käyttää kolmen tyyppistä tallennusta: väliaikaista, pysyvää

sekä rajoittamatonta. Oletuksena käytössä on väliaikainen tallennus, jota varten ei tarvitse pyytää käyttäjältä mitään erityisiä lupia. (chrome.com.)

Väliaikainen tallennus on jaettu kaikkien selaimen käyttämien sovellusten ja rajapintojen välillä (poislukien Local- ja SessionStorage). Jaetun kovalevyn määrä on korkeintaan 1/3 vapaana olevasta kovalevytilasta, väliaikaisen tallennuksen jo käyttämä tila huomioidaan laskuissa. Yksittäinen sovellus tai rajapinta voi saada korkeintaan 20% jaetusta kovalevystä käyttöönsä, tilan loppuessa vanhimman (least recent) käytetyn alkuperän tallentama tieto poistetaan kokonaisuudessaan. (chrome.com.) Incognito –tilaa käytettäessä väliaikaisen tallennuksen kanssa, tuhotaan rajapinnoilla tallennettu tieto selaimen sulkemisen jälkeen.

Pysyvä tallennus pysyy selaimen käytössä siihen asti, kunnes käyttäjä poistaa sen. Sovelluksen on mahdollista käyttää sille määritelty määrä kovalevytilaa, mutta tämä vaatii lupaa käyttäjältä. Toistaiseksi tallennustapa on vain File System rajapinnan käytössä. (chrome.com.)

Rajoittamaton tallennus voi tallentaa kovalevylle rajattomasti tietoa tarpeiden mukaan, mutta on käytettävissä vain Chromeen tehdyissä sovelluksissa ja lisäosissa. Käyttäjän on asennusvaiheessa sallittava kovalevyn vapaa käyttö. (chrome.com.)

5 Tutkimuksen tulokset

Kappale sisältää rajapintakohtaisen esittelyn, vastaan tulleet ongelmat ja rajoitteet kirjasto-demoa tehdessä ja lopuksi nopeustestit CRUD-operaatioista.

5.1 Web SQL Database

Rajapintaan oli helppo päästä sisälle, kun relaatiotietokannat ja SQL olivat ennestään jo tuttuja. Kirjasto Demo –sovelluksessa luodaan käyttäjä, kirja ja käyttäjäKirja –taulut, jotka on kuvattu alla (taulukko 1, taulukko 2, taulukko 3).

Taulukko 1. Käyttäjätaulun kaavio

Nimi	Tyyppi	Null	Avain	Lisätietoa
käyttäjäid	integer	ei	pää	auto increment
etunimi	teksti	ei	-	-
sukunimi	teksti	ei	-	-
Osoite	teksti	ei	-	-
sähköposti	teksti	ei	-	-
kirjastokortti	teksti	ei	-	uniikki
päivitetty	integer	ei	-	oletus: 0

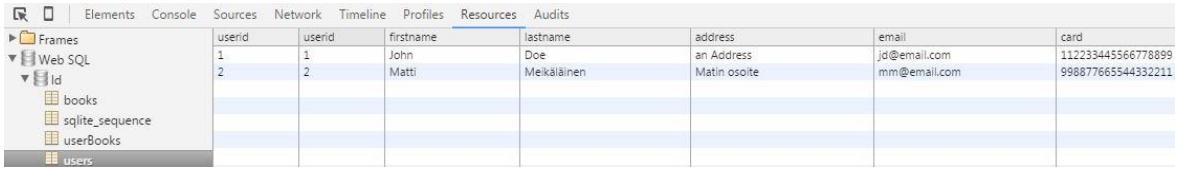
Taulukko 2. Kirjataulun kaavio

Nimi	Tyyppi	Null	Avain	Lisätietoa
kirjaid	integer	ei	pää	auto increment
nimi	teksti	ei	-	-
tekijä	teksti	ei	-	-
genre	teksti	ei	-	-
isbn	teksti	ei	-	uniikki
päivitetty	integer	ei	-	oletus: 0

Taulukko 3. käyttäjäKirjataulun kaavio

Nimi	Tyyppi	Null	Avain	Lisätietoa
kkid	integer	ei	pää	auto increment
käyttäjä	integer	ei	viite	käyttäjätaulun käyttäjid
kirja	integer	ei	viite	kirjataulun kirjaid

Seuraavassa kuviossa nähdään tietokannan rakenne ja sisältö Google Chrome –selaimen kehittäjän työkaluista katsottuna (Kuvio 7).



Kuvio 7. Taulun rakenne ja sisältö Chromesta katsottuna.

Kun tietoa halutaan tallentaa, tarkistetaan ensin tietokantojen olemassaolo, jonka jälkeen aloitetaan itse tapahtuma. Sovelluksen tapahtumien rakenne on: kysely, taulukko injektioita vastaan, onnistuneen- ja epäonnistuneen kyselyn käsittely, epäonnistuneen- ja onnistuneen tapahtuman käsittely. Alla olevassa kuviossa on pätkä demosovelluksessa käytetystä koodista (Kuvio 8).

```
this.transaction(function (tx){//Tapahtuma alkaa
    tx.executeSql(('INSERT INTO users (firstname, lastname, address, email, card) VALUES (?,?,?,?,?)'), //SQL kysely
    [fName, lName, address, email, libraryCard],//injektion esto, kysymysmerkit korvataan taulukon arvoilla.
    function(tx, result){//kysely onnistuu
        if(result){
            console.log(result);
            document.getElementById("fNameField").value="";
            document.getElementById("lNameField").value="";
            document.getElementById("addressField").value="";
            document.getElementById("emailField").value="";
            document.getElementById("cardField").value="";
            document.getElementById("newUserStatus").innerHTML="<b>Saved</b>";
        }
    },
    function(err){//kysely epäonnistuu
        console.log("Database error:");
        console.dir(err);
    });
},
function(err){//Tapahtuma epäonnistuu
    console.log("Database error:");
    console.dir(err);
},
function(){//Tapahtuma onnistuu
    console.log("Transaction finished");
});
```

Kuvio 8. Koodipätkä käyttäjän lisäämistapahtumasta.

Tiedon tallennuksen kulku on seuraava: Käyttäjän painaessa Save –nappia addUser –sivulla, kutsutaan newUser –funktiota. Funktio tarkistaa ensin onko kyseessä tests –sivun testi (ei ole), jonka jälkeen tarkistetaan, että kaikki kentät on täytetty. Jos kaikki tiedot löytyvät, suoritetaan insert into –kysely. SQL injektioiden estämiseksi, arvot lisätään dynaamisesti taulukosta niille varattuihin kohtiin (?). Tapahtuman onnistuessa, ilmoitetaan siitä käyttäjälle ja tyhjennetään kaikki kentät.

Arvojen dynaaminen lisääminen tuotti hieman ongelmia tilanteissa, joissa tallennettavien arvojen tarkka lukumäärä ei ollut etukäteen tiedossa. Sovelluksessa kirjojen lainaaminen

ja palauttaminen tuottaa tilanteen, jossa ei voida ennalta määrittää käsiteltävien arvojen lukumäärää. Asiakas voi lainata tai palauttaa n määrän kirjoja kerralla, jolloin vaihtoehtoina ovat joko useamman kyselyn suorittaminen tai luoda tarvittava määrä kysymysmerkkejä silmukassa, joka laskee kirjojen määrän. Päädyin sovelluksessani jälkimmäiseen ratkaisuun.

Kyselylauseelle on määritelty jokin maksimipituus, joka selvisi ajaessani lisäystestiä 100 000 käyttäjällä JSON –tiedostosta tietokantaan. Testatessani pienemmillä määrillä, kysely meni onnistuneesti läpi. Muutin lisäystestiä niin, että käyttäjät lisätään yksitellen silmukan läpi tietokantaan.

Tiedon hakeminen tietokannasta toimii rakenteellisesti samalla tavalla, kuin tiedon lisäyksessä, kuten alla olevasta kuviosta on nähtävissä (Kuvio 9). Käyttäjälistan hakutapahtuma suoritetaan sovelluksen käyttäjän siirtyessä listUsers –sivulle. getUserList –funktio ajetaan sivun latautuessa, jonka jälkeen sivuun lisätään lista käyttäjistä jos niitä on tallennettuna.

```
this.readTransaction(function (tx){ //Lukutapahtuma alkaa
    tx.executeSql('SELECT userid, firstname, lastname from users', [], //Kysely, joka palauttaa kaikki käyttäjät.
    //Taulukko injektion estoja varten on oltava mukana vaikka se olisi tyhjä eikä tilanteessa tarpeellinen.
    function(tx, result){// Kyselyn onnistuessa.
        if(result){
            for(var i=0;i<result.rows.length;i++){
                results+=(i+1)+". <a href='user.html?key="+result.rows.item(i).userid+"'>"
                +result.rows.item(i).firstname+" "+result.rows.item(i).lastname+"</a><br />";
            }
            console.log("User list done!");
            document.getElementById("userList").innerHTML=results;
        }
    },
    function(err){// Kyselyn epäonnistuessa
        console.log("Database error:");
        console.dir(err);
    });
},
function(err){// Tapahtuman epäonnistuessa
    console.log("Database error:");
    console.dir(err);
},
function(){//Tapahtuman onnistuessa
    console.log("Transaction finished");
});
```

Kuvio 9. Koodipätkä kaikkien käyttäjien hakemisesta lukutapahtumassa.

Tietoa päivitettäessä user –sivulla, asetetaan oletusarvoiksi vanhat tiedot, joita sovelluksen käyttäjä voi muuttaa tarpeidensa mukaan. Kuten aiemmin, itse tietokannan käsittelyssä ei muutu muu, kuin suoritettu kysely. Alla olevassa kuviossa (Kuvio 10) näkyy JavaScript funktio, kun päivitys suoritetaan käyttäjän painaessa Edit -nappia.

```
function modifyUserById(id){
    var fName=document.getElementById("editFname").value;
    var lName=document.getElementById("editLname").value;
    var address=document.getElementById("editAddress").value;
    var email=document.getElementById("editEmail").value;
    var card=document.getElementById("cardId").value;
    if(fName&&lName&&address&&email&&card&&id){
        connect(function(){ //Tietokantayhteyden avaus
            if(this){
                this.transaction(function (tx){ //Tapahtuma
                    tx.executeSql('UPDATE users SET firstname=?, lastname=?, address=?, email=? WHERE userid=?',
                    [fName, lName, address, email, id],
                    function(tx, result){
                        if(result){
                            document.getElementById("userEditStatus").innerHTML="<b>User modified</b>";
                        }else{
                            console.log("Update failed");
                        }
                    },
                    function(err){
                        console.log("Database error:");
                        console.dir(err);
                    });
                });
            }
        });
    }else{
        document.getElementById("userEditStatus").innerHTML="<b>All fields aren't filled</b>";
    }
}
```

Kuvio 10. Koodi käyttäjätietojen päivittämisestä.

Tietoa poistettaessa (tässä tapauksessa käyttäjää) samaisella user –sivulla, painetaan Delete –nappia. Napin painaminen kutsuu removeUserById –funktiota, joka on nähtävissä alla olevassa kuviossa (Kuvio 11).

```
function removeUserById(id){
    if(id){
        connect(function(){
            if(this){
                this.transaction(function (tx){
                    tx.executeSql('DELETE FROM users WHERE userid=?', [id],
                    function(tx, result){
                        if(result){
                            //Siirrytään takaisin käyttäjien listaussivulle.
                            window.location.replace("listUsers.html");
                        }else{
                            console.log("User not found");
                        }
                    },
                    function(err){
                        console.log("Database error:");
                        console.dir(err);
                    });
                });
            }
        });
    }
}
```

Kuvio 11. Koodi käyttäjän poistamisesta.

5.2 Indexed Database

Indexed Databaseen perehtyminen vei hieman pidempään, tutustuminen MongoDB:n toimintaan sekä aikaisempi kokemus JSON:ista helpotti hahmottamaan tietokannan rakennetta ja toimintaa. Tiedot tallennetaan olioihin, jotka tukevat yksinkertaisia avain-arvo pareja sekä taulukkoja. Kirjasto Demo –sovelluksessa luodaan käyttäjät- ja kirjat –oliovarastot, nämä ovat kuvattuna alla (Taulukko 4, Taulukko 5).

Taulukko 4. Käyttäjät –oliovarasto

Avain (1-n)	**Arvo (arvo.etunimi, arvo.sukunimi…)**
	etunimi
	sukunimi
	osoite
	sähköposti
	kirjastokortti
	päivitetty

Taulukko 5. Kirjat –oliovarasto

Avain (1-n)	**Arvo (arvo.nimi, arvo.tekijä…)**
	nimi
	tekijä
	genre
	isbn
	päivitetty

Koska Indexed Databasessa ei ole tarvetta määritellä kaaviota etukäteen (eikä rakennetta ole lyöty lukkoon), voidaan jättää ”välitaulu” pois ja tallentaa käyttäjän lainat suoraan käyttäjätauluun. Alla kuvaus käyttäjät -oliovarastosta, johon on lisätty tietoa lainatuista kirjoista (Taulukko 6).

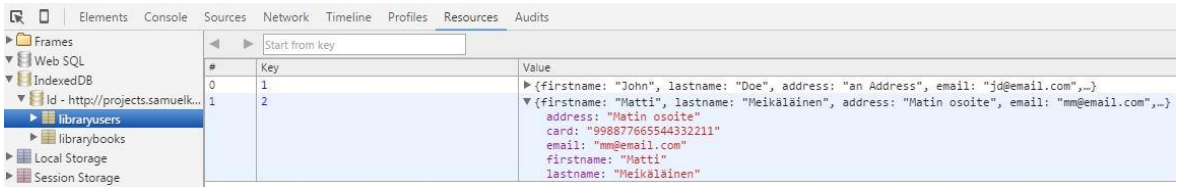
Taulukko 6. Käyttäjät –oliovarasto lainatuilla kirjoilla

Avain (1-n)	**Arvo (arvo.etunimi, arvo.sukunimi…)**
	etunimi
	sukunimi
	osoite
	sähköposti
	kirjastokortti

	päivitetty
	kirjat[kirjaid, kirjaid, kirjaid…]

Kirjojen id:t on tallennettu taulukkoon, joka on lisätty sellaisenaan oliovarastoon.

Seuraavassa kuviossa nähdään oliovaraston rakenne ja sisältö Google Chrome –selaimen kehittäjän työkaluista katsottuna (Kuvio 12).



Kuvio 12. Oliovaraston rakenne ja sisältö Chromesta katsottuna.

Tietoa tallennettaessa, tarkistetaan ensin, onko tietokannan uusin versio käytössä (sama jos tietokantaa ei ole luotu). Jos oliovarastoja ei ole olemassa, ne luodaan ja halutut indeksit määritellään. Itse tapahtuma alkaa sovelluksessa sen määrittelyllä ja oliovaraston avaamisella. Kysely ajetaan, jonka jälkeen kuvataan tapahtuman onnistuminen, kyselyn epäonnistuminen ja –onnistuminen. Alla olevassa kuviossa on pätkä demosovelluksessa käytetystä koodista (Kuvio 13).

```
var user={firstname:fName, lastname:lName, address:address, email:email, card:libraryCard}//luodaan taulukko tallennettavasta tiedosta.
var transaction=this.transaction(["libraryusers"],"readwrite");//Luodaan luku & kirjoitus tapahtuma "libraryusers" -oliovarastolle.
var objstore=transaction.objectStore("libraryusers");//Avataan itse oliovarasto.
var query=objstore.add(user);//Luodaan luonti -kysely.
transaction.oncomplete=function(){//Suoritetaan tapahtuman valmistuessa.
    console.log("User saved");
    document.getElementById("fNameField").value="";
    document.getElementById("lNameField").value="";
    document.getElementById("addressField").value="";
    document.getElementById("emailField").value="";
    document.getElementById("cardField").value="";
    document.getElementById("newUserStatus").innerHTML="<b>Saved</b>";
}
query.onerror=function(err){//Suoritetaan kyselyn epäonnistuessa.
    console.log("Error: ", err.target.error.name);
}
query.onsuccess=function(){//Suoritetaan kyselyn onnistuessa.
    console.log("Saving user");
}
```

Kuvio 13. Koodipätkä käyttäjän lisäämistapahtumasta.

Tiedon tallennuksen kulku on seuraava: Käyttäjän painaessa Save –nappia addUser –sivulla, kutsutaan newUser –funktiota. Funktio tarkistaa ensin onko kyseessä tests –sivun testi (ei ole), jonka jälkeen tarkistetaan, että kaikki kentät on täytetty. Jos kaikki tiedot löytyvät, tallennetaan ne käyttäjä –olioon. Tapahtuma avataan libraryusers -oliovarastoon ja lisäys –kysely suoritetaan käyttäjä –oliolle. Tapahtuman onnistuessa, ilmoitetaan siitä käyttäjälle ja tyhjennetään kaikki kentät.

Tietoa haettaessa listUsers –sivulle, luodaan tapahtuma oliovarastoon lukuoikeuksilla, jonka jälkeen tapahtumalle avataan osoitin käymään läpi tietokantaan tallennettuja käyttäjälistoja. Kuviossa (Kuvio 14) näkyy JavaScriptin käyttäjälistan haku funktio.

```
function getUserList(){
    connect(function(){
        if(this){
            var result="";
            var count=0;
            var transaction=this.transaction(["libraryusers"],"readonly");
            // Alla avataan osoitin iteroimaan löydettyjä käyttäjiä
            var query=transaction.objectStore("libraryusers").openCursor();
            transaction.oncomplete=function(){
                document.getElementById("userList").innerHTML=result;
            }
            query.onsuccess=function(res){
                var cursor=res.target.result;
                if(cursor){
                    count++;
                    result+=count+". <a href='user.html?key="+cursor.key+"'>"
                    +cursor.value.firstname+" "+cursor.value.lastname+"</a><br />";
                    cursor.continue(); // Osoitin jatkaa seuraavalle kierrokselle
                }
            }
        } else{
            console.log("Database didn't load");
        }
    });
}
```

Kuvio 14. Koodi käyttäjälistan hakemisesta.

Tiettyjen arvojen etsiminen on Indexed Databasessa hieman rajoittunut. Web SQL Databasessa, kuten SQL -kielessä ylipäätänsä on tarkkojen arvojen ja matemaattisten vertailujen lisäksi mahdollista käyttää like –operaattoria. Tämä mahdollistaa jokerimerkkien käytön SQL hauissa. Indexed Databasessa vuorostaan voidaan indeksiä käyttäen joko hakea täsmällistä arvoa kirjainkokoa myöten, kaikki arvot halutun arvon ylä- tai alapuolelta tai arvoja kahden arvon välistä. Esimerkiksi MongoDBssa voi SQL:n tavoin myös käyttää täsmällisemmän haun lisäksi matemaattisia vertailuja sekä ja- ja tai –operaattoreita.

Päivitys user –sivulla tapahtuu samoin, kuin Web SQL Databasella. Kun Edit –nappia on painettu, avataan tietokantayhteys ja suoritetaan päivitystapahtuma valitulle käyttäjälle (olettaen, että kaikki tiedot on annettuna). Alla olevassa kuviossa (Kuvio 15) koodia päivitystapahtumasta.

```
connect(function(){ //Tietokantayhteyden avaus
    if(this){
        var transaction=this.transaction(["libraryusers"],"readwrite");
        var objstore=transaction.objectStore("libraryusers");
        //put -komentoa käytetään uusiin sekä päivityksiin. add -komento toimii vain uusiin.
        var query=objstore.put(user, id); //Päivitettäessä tarvitaan tallennettavan tiedon lisäksi listan id.
        transaction.oncomplete=function(){
            document.getElementById("userEditStatus").innerHTML="<b>User modified</b>";
        }
        query.onerror=function(err){
            console.log("Error: ", err.target.error.name);
            document.getElementById("userEditStatus").innerHTML="<b>There was an error...</b>";
        }
        query.onsuccess=function(){
            console.log("Modifying user");
        }
    } else{
        console.log("Database didn't load");
    }
});
```

Kuvio 15. Koodipätkä käyttäjätietojen päivittämisestä.

Käyttäjän poisto user –sivulla painettaessa Delete –nappia, tapahtuu removeUserById –funktiossa avaamalla tietokantaan luku- ja kirjoitustapahtuma ja suorittamalla delete –komento annetulle listan id:lle. Alla kuvio (Kuvio 16) käyttäjän poistosta.

```
function removeUserById(id){
    if(id){
        connect(function(){
            if(this){
                var transaction=this.transaction(["libraryusers"],"readwrite");
                var objstore=transaction.objectStore("libraryusers");
                var query=objstore.delete(Number(id));
                transaction.oncomplete=function(){
                    //Siirrytään takaisin käyttäjien listaussivulle.
                    window.location.replace("listUsers.html");
                }
                query.onerror=function(err){
                    console.log("Error: ", err.target.error.name);
                }
                query.onsuccess=function(){
                    console.log("Deleting user");
                }
            } else{
                console.log("DB didn't load");
            }
        });
    }
}
```

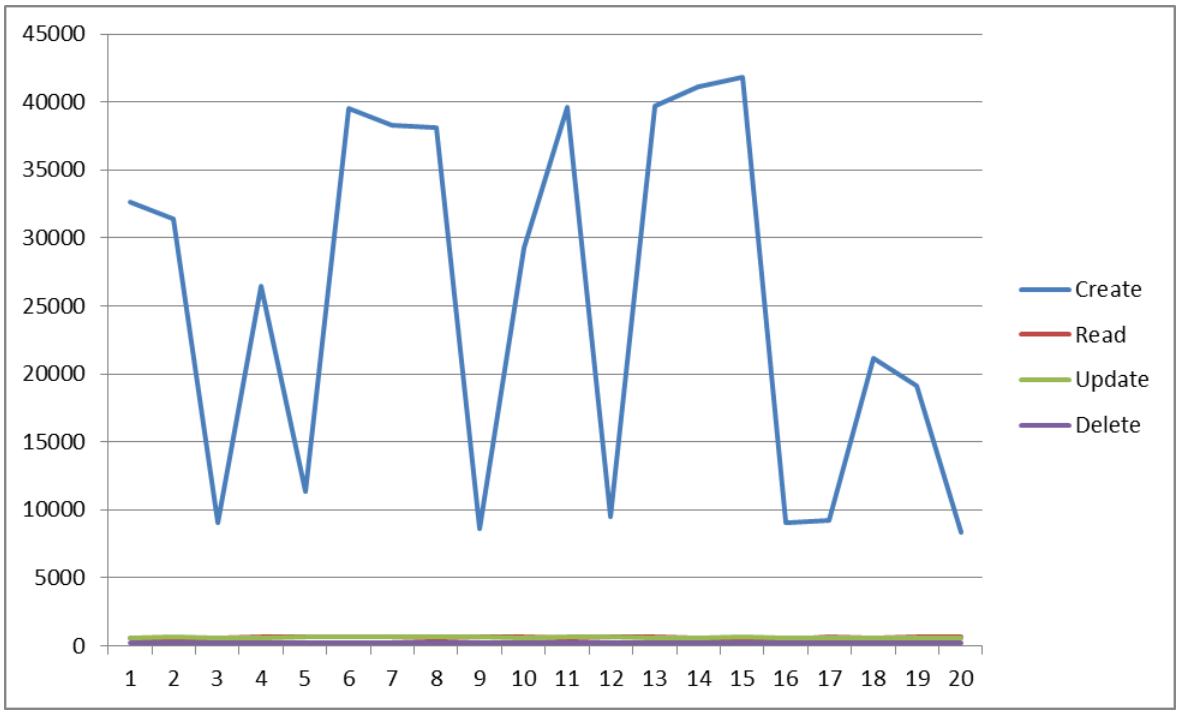
Kuva 16. Koodi käyttäjän poistamisesta.

5.3 CRUD nopeustestit

Testien suoritushetkellä käytettiin Google Chromen versiota 47.0.2526.106 m. Testejä ajettiin 20 kierrosta, jonka jälkeen niitä ei enää voitu suorittaa Indexed Database-rajapinnalla Chrome-selaimessa ilman jatkuvaa uudelleenkäynnistelyä. Mahdollinen syy

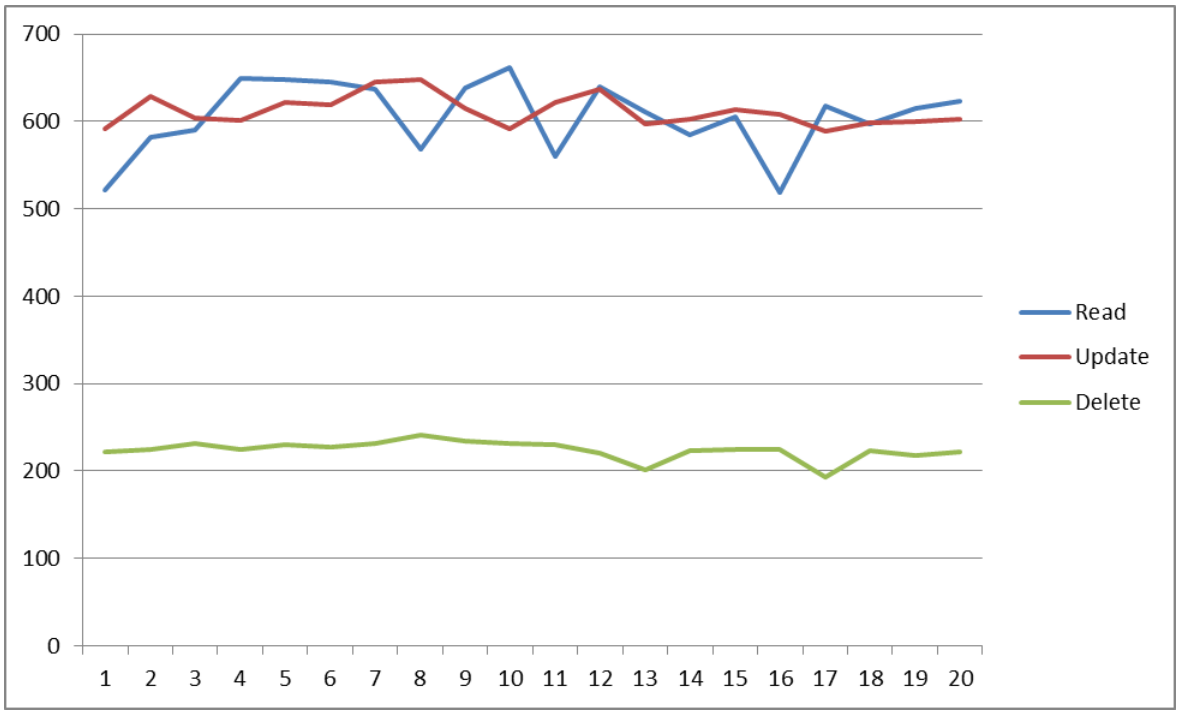
tähän voi olla Chromen taustalla suorittama selaimen päivitys. Chromen versionumeron tulin katsoneeksi vasta testien jälkeen. Tarkistaessani olinko tehnyt koodissani virheitä, totesin Firefoxin pystyvän edelleen ajamaan useamman testikierroksen ilman ongelmia.

Web SQL Databasen nopeustestin tulokset millisekunteina diagrammissa (Kuvio 17). Yksittäiset tulokset löytyvät liitteistä (liite 1).



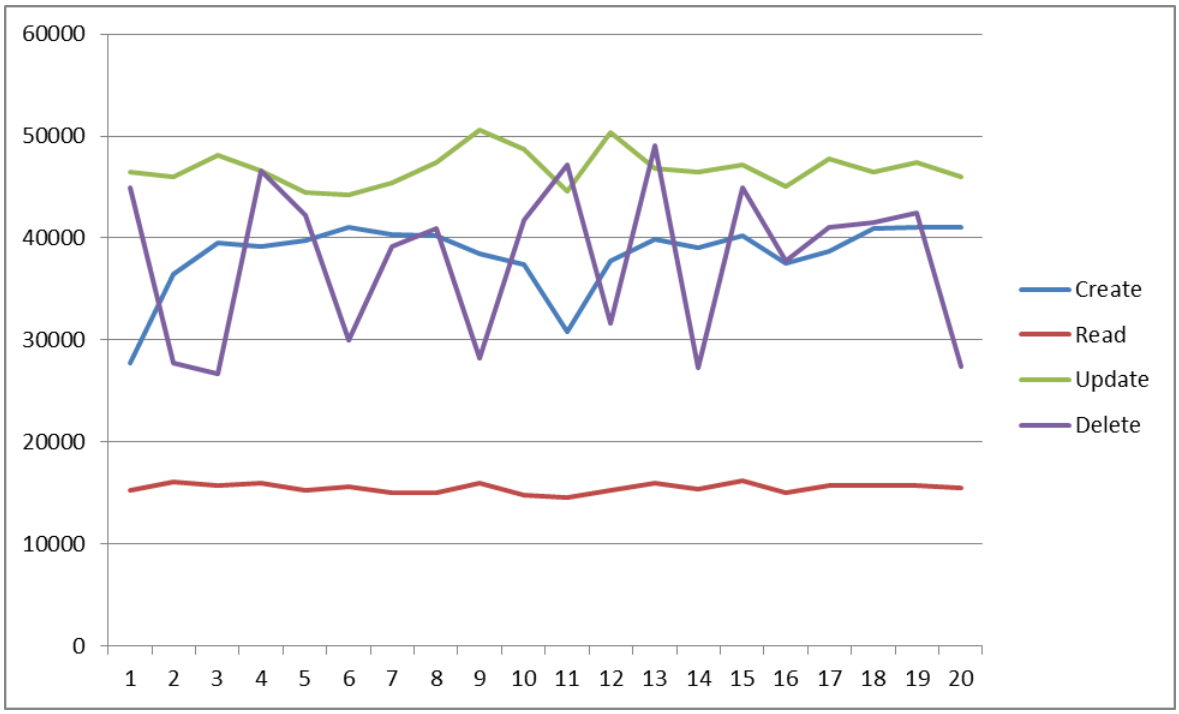
Kuvio 17. Web SQL Databasen CRUD-testien ajat.

Koska Create testi vei muihin testeihin verrattuna paljon pidempään, alla kuvio testeistä ilman Createa (Kuvio 18).



Kuvio 18. Nopeustestit ilman Create testiä.

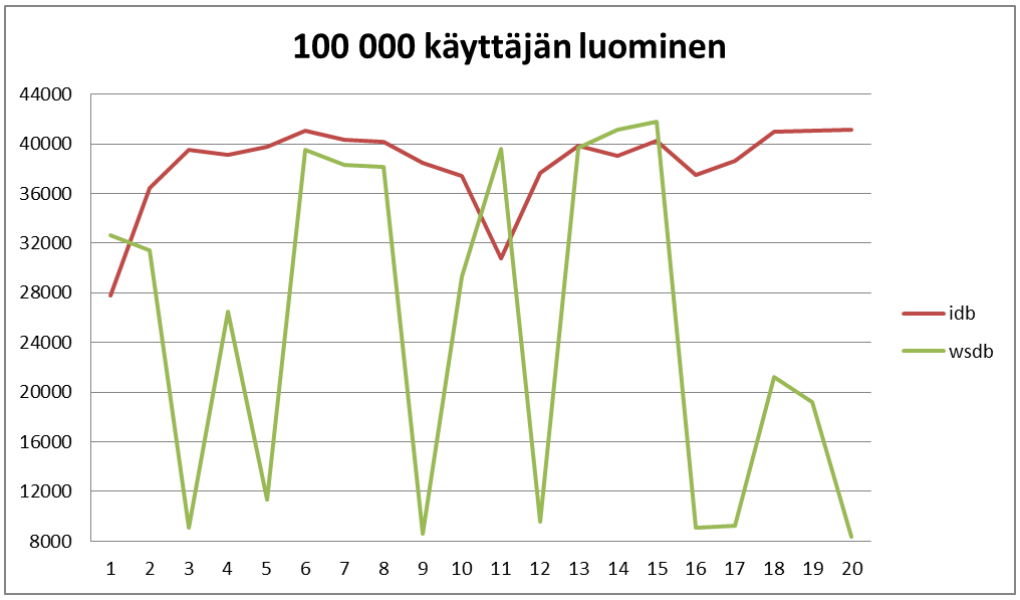
Indexed Databasen nopeustestin tulokset millisekunteina diagrammissa (Kuvio 19). Yksittäiset tulokset löytyvät liitteistä (liite 2).



Kuvio 19. Indexed Databasen CRUD-testien ajat.

Create-testi poislukien, Web SQL Database suoriutui kaikista operaatioista huomattavasti Indexed databasea nopeammin. Kuten alla olevasta kuviosta käy ilmi (Kuvio 20), Create testissä Web SQL Database suoriutui parhaimmillaan alle kymmenessä sekunnissa, kun

taas Indexed Databasella meni lähes puoli minuuttia. Suurin kulunut aika oli yllättäen myös Web SQL Databasella, joka oli vajaan sekunnin suurempi, kuin Indexed Databasen suurin kulunut aika.



Kuvio 20. Indexed Databasen sekä Web SQL Databasen Create testi millisekunteina.

Web SQL Databasen omia testejä katsoessa voisi spekuloida, että Create-testin viemään aikaan voisi vaikuttaa JSON-tiedoston lukeminen. Indexed Databasen testejä katsoessa, Create-testin viemä aika ei eroa samalla tavalla suhteessa muihin testeihin. Tämän lisäksi Indexed Databasen Create-testin nopeusvaihtelut ovat reilusti maltillisempia Web SQL Databasen testiin nähden.

6 Pohdinta

Tässä kappaleessa pohditaan rajapintojen hyötyjä ja käyttökohteita.

Näkisin suurimman hyödyn rajapintojen käytössä erinäisissä web- ja mobiilisovelluksissa jotka toimivat paikallisesti käyttäjän laitteilla. Websovellukset eivät vaadi ylimääräisiä ympäristöjen asennuksia ja mobiili puolella esimerkiksi Apache Cordovalla on mahdollista toteuttaa sovelluksia useille eri käyttöjärjestelmille HTML:n, CSS:n sekä JavaScriptin avulla. Toinen hyöty voisi olla erinäisten prototyyppien tai proof of concept projektien nopeassa kehittämisessä juurikin minimaalisten ympäristövaatimusten takia (yleisimmät selaimet).

En toistaiseksi näe rajapinnoilla paljoa käyttöä edellä mainittujen esimerkkien ulkopuolella. Web Storage vaikuttaisi paljon todennäköisemmältä vaihtoehdolta korvaamaan evästeet, jos niistä halutaan luopua. Rajapintoja voitaisiin käyttää myös vähämerkityksellisen tiedon tallentamiseen internetpalveluissa esimerkiksi hakujen nopeuttamiseksi mm. karttasovelluksessa tai tallentamaan käyttäjän preferenssejä, kuten sivuston teema-asetuksia. Koska selaimen ei enää tarvitse hakea kaikkea tietoa jatkuvasti palvelimelta, vähentynee tämän käyttökuorma huomattavalla tasolla.

Vaikka Web SQL Database oli lähes poikkeuksetta selkeästi Indexed Databasea nopeampi kaikissa mitatuissa testeissä (SQL on myös todennäköisesti tutumpi kieli monelle), voin antaa rajapinnasta vain harkitun tilannekohtaisen suosituksen. Koska Web SQL Database ei ole enää aktiivisessa kehityksessä, on mahdollista, että selaimet pudottavat tuen tälle rajapinnalle. Indexed Database on W3C:n luokittelema virallinen verkon standardi. Se oli ainakin minulle jokseenkin helppo omaksua ja varmemman tuen puitteissa soveltuu paremmin käytettäväksi mobiilisovelluksiin.

7 Yhteenveto

Opinnäytetyön tavoitteena oli ottaa selvää kuinka käyttäjän puolella tapahtuva verkkosivujen tiedon varastointi toimii Indexed Database- sekä Web SQL Database–rajapinnoilla ja tutustua mahdollisiin käyttöönoton vaatimuksiin. Rajapinnat ovat puhdasta JavaScriptia ja toimivat niitä tukevissa selaimissa ilman ylimääräisiä kirjastoja tai ohjelmistokehyksiä. Tutkimusosuudessa käytiin läpi rajapintojen syntaksia ja toimintaa käytännön projektin avulla.

Teoriataustassa käytiin läpi lyhyesti tietokannan historiaa, fyysisestä mediasta digitaaliseen, hajanaisempaan ohjelmakohtaiseen tiedon säilyttämiseen. Tästä edelleen yhtenäisempään tiedon tallennusratkaisuun, tietokantaan. Tietokanta -teoria on jaettu relaatiotietokantoihin sekä NoSQL tietokantoihin, joihin Web SQL Database sekä Indexed Database vastaavasti kuuluvat.

Tutkimuksessa kerrotaan rajapintojen käyttöönotosta, joka ei vaadi tuetun selaimen lisäksi muuta, kuin JavaScript osaamista ja itse rajapinnan syntaksiin tutustumista. Molemmista rajapinnoista käydään läpi esimerkki eri CRUD –operaatioista. Rajapintojen käyttö ei ollut täysin ongelmatonta. Demon luontivaiheessa jouduin ratkomaan molemmissa rajapinnoissa ilmenneitä toiminnallisuuden ongelmakohtia, joista merkittävimmät olivat listattuna. Demon CRUD –operaatioiden nopeuksien mittauksessa ilmeni, että Web SQL Database oli lähes poikkeuksetta nopeampi. Tästä huolimatta tätä rajapintaa oli hankala suositella pidempiin projekteihin, tuen mahdollisen päättymisen takia.

8 Oppimiskokemukset

Olen opinnäytetyön aikana kehittänyt osaamistani relaatiotietokannoista, niiden historiasta, käytöstä ja eritoten oikeiden termien käytöstä. Aiemmin olin käyttänyt SQL:n termejä oikeiden relaatiotietokannan termien sijaan. Ennen opinnäytetyön aloittamista oli NoSQL minulle tuttu vain terminä ja tiesin MongoDB:n olevan yksi niiden edustajista. MongoDB:stä minulla oli sellainen käsitys, että tallennetulla tiedolla ei ole yhteyksiä toistensa välillä vaan hakuja olisi suoritettava useampi, jotta saataisiin sama tieto relaatiotietokantoihin verrattuna.

Projektin suunnitteluvaiheessa olin hyvin kiintynyt ajatukseen tietokantoihin liittyvästä aiheesta. Ensimmäinen ajatukseni oli vertailla Relaatiotietokantaa ja NoSQL tietokantaa keskenään (MySQL & MongoDB). JavaScriptistä ja Json:ista aiemmilta kursseiltani innostuneena, päädyin lopulta tiedon varastoinnin ja muokkaamisen mahdollisuuksien selvittämiseen front-endissä. Web Storagen kautta tutustuin Indexed Databaseen sekä Web SQL Databaseen, joita halusin lähteä tutkimaan opinnäytetyöhöni.

Teoriataustaa etsiessäni tulin todenneeksi, että rajapintojen käyttökohteet vaikuttavat hyvin rajallisilta. Tämä varmaankin selittää miksi materiaalia ei tuntunut löytyvän W3C:n spesifikaatioiden, pintapuolisten esittelyjen ja arvostelujen sekä koodiesimerkkien lisäksi. Kattavan teorian saamiseksi, kävin läpi teoriaa relaatio- ja NoSQL tietokannoista.

Relaatiotietokannoista ja SQL:stä minulla oli jo ennestään kokemusta enimmäkseen MySQL:n käytöstä. NoSQL:ään tutustuessani selvisikin, että olin jo tavallaan käyttänyt avain-arvo- sekä dokumenttitietokantoja ennestään. Avain-arvo tietokannat toimivat pääsääntöisesti samalla tavalla, kuin Web Storage (Local- & SessionStoragen avain-arvo parit) ja dokumenttitietokannat taas vuorostaan tuntuivat vastaavan rakenteeltaan JSON–tiedostoja. Sarakepohjainen tietokanta oli jotenkin löyhästi rinnastettavissa relaatiotietokannan sekä dokumenttitietokannan hybridiin. Graph tietokantaa en pystynyt rinnastamaan oikein mihinkään aikaisempaan kokemukseen ja siihen tutustuminen vaatikin eniten aikaa NoSQL tietokannoista.

Web SQL- ja Indexed Databaseen tutustumisen lisäksi tulin tutkimusosuudessa tehneeksi molempia rajapintoja käyttävän demon, jolla pystyin eri toimintojen testaamisen lisäksi testaamaan ja kellottamaan CRUD –operaatioiden nopeuksia isoissa tapahtumamäärissä.

Lähteet

bsonspec.org. BSON. Luettavissa: http://www.bsonspec.org/. Luettu: 16.9.2015.

chrome.com. Managing HTML5 Offline Storage. Luettavissa: https://developer.chrome.com/apps/offline_storage/. Luettu: 04.01.2016.

Connolly, T., Begg, C. 2015. Database Systems A Practical Approach to Design, Implementation, and Management. Sixth edition. Pearson Education Limited. Harlow.

Fowler, M. 2012. NosqlDefinition. Luettavissa: http://www.martinfowler.com/bliki/NosqlDefinition.html. Luettu: 24.8.2015.

Harkins, S. 2003. Relational Databases: Defining relationships between database tables. Luettavissa: http://www.techrepublic.com/article/relational-databases-defining-relationships-between-database-tables/. Luettu: 7.9.2015.

Hickson, I. 2009a. Web Storage. Luettavissa: http://www.w3.org/TR/2009/WD-webstorage-20090423/. Luettu: 8.7.2015.

Hickson, I. 2009b. Web Database. Luettavissa: www.w3.org/TR/2009/WD-webdatabase-20090910/. Luettu: 8.7.2015.

Hickson, I. 2015. Web Storage (Second Edition). Luettavissa: http://www.w3.org/TR/2015/CR-webstorage-20150609/. Luettu: 29.6.2015.

Hickson, I. 2010. Web SQL Database. Luettavissa: http://www.w3.org/TR/2010/NOTE-webdatabase-20101118/. Luettu: 24.6.2015.

json.org. Introducing JSON. Luettavissa: http://www.json.org/. Luettu: 16.9.2015.

Kontiomaa, S. 2016. IDB-WSDB-Demo. Luettavissa: https://gitlab.com/Kontiomaa/IDB-WSDB-Demo/. Luettu: 20.3.2016.

Laiho, M. 1990. SQL. 1. – 2. painos. Valtion painatuskeskus. Helsinki.

Aller, M, 2010. HTML5 Web SQL Database – Intro to Versioning and Migrations. Luettavissa: http://blog.maxaller.name/2010/03/html5-web-sql-database-intro-to-versioning-and-migrations/. Luettu 2.12.2015.

McCreary, D., Kelly, A. 2013. Making Sense of NoSQL: A guide for managers and the rest of us. 1 edition. Manning Publications Co. Shelter Island, New York.

Mehta, N., Sicking, J., Graff, E., Popescu, A., Orlow, J. & Bell, J. 2015. Indexed Database API. Luettavissa: http://www.w3.org/TR/2015/REC-IndexedDB-20150108/. Luettu: 24.6.2015.

Mehta, N. 2009. WebSimpleDB API. Luettavissa: www.w3.org/TR/2009/WD-WebSimpleDB-20090929/. Luettu: 8.7.2015.

mongodb.com. JSON and BSON. Luettavissa: https://www.mongodb.com/json-and-bson/. Luettu: 16.9.2015.

Neubauer, P. 2010. Graph Databases, NOSQL and Neo4j. Luettavissa: http://www.infoq.com/articles/graph-nosql-neo4j/. Luettu: 08.09.2015.

nosqlguide 2014a. GUIDE TO WIDE-COLUMN STORES – NOSQL EXPLAINED. Luettavissa: http://nosqlguide.com/column-store/nosql-databases-explained-wide-column-stores/. Luettu: 11.8.2015

nosqlguide 2014b. GUIDE TO GRAPH DATABASES – NOSQL EXPLAINED. Luettavissa: http://nosqlguide.com/graph-database/nosql-databases-explained-graph-databases/. Luettu: 11.8.2015

Quin, L. 2015. Extensible Markup Language (XML). Luettavissa: http://www.w3.org/XML/. Luettu: 16.9.2015.

Sadalage, P., Fowler M. 2012. NoSQL distilled: A Brief Guide to the Emerging World of Polyglot Persistence. First printing. Pearson Education, Inc. Crawfordsville, Indiana.

The World Wide Web Consortium 2015a. About W3C. Luettavissa: www.w3.org/Consortium/. Luettu: 26.6.2015.

The World Wide Web Consortium 2015b. Standards. Luettavissa: www.w3.org/standards/. Luettu: 18.11.2015.
Ullman, J., Widom, J. 1997. A First Course in Database Systems. 1st edition. Prentice-Hall, Inc. New Jersey.

W3Schools 1999. HTML5 Local Storage. Luettavissa: www.w3schools.com/html/html5_webstorage.asp. Luettu: 7.7.2015.

Zakas, N. 2009. http cookies explained. Luettavissa: www.nczonline.net/blog/2009/05/05/http-cookies-explained/. Luettu 29.6.2015

Liitteet

Liite 1. Indexed Databasen Crud-testin ajat millisekunteina

100k test	Create	Read	Update	Delete
1	27780,45	15203,065	46411,68	44919,315
2	36406,555	16094,2449	45953,375	27716,655
3	39547,42	15678,3749	48101,64	26701,219
4	39124	15908,415	46610,5199	46598,255
5	39754,315	15262,795	44489,755	42191,15
6	41065,095	15592,5199	44186,715	29963,625
7	40306,635	14968,24	45438,5799	39109,17
8	40191,18	15017,0599	47377,145	40920,23
9	38475,26	15905,39	50618,1899	28203,5099
10	37449,325	14743,4099	48738,35	41751,89
11	30742,125	14543,485	44587,75	47180,8199
12	37698,61	15265,99	50304,215	31581,63
13	39866,81	15952,415	46823,57	49006,9599
14	39067,615	15396,075	46459,965	27274,0499
15	40256,85	16135,575	47172,2099	44944,24
16	37504,155	15015,815	45050,78	37719,18
17	38649,97	15707,2699	47777,31	40989,2099
18	40970,88	15693,54	46467,9099	41509,155
19	41023,775	15745,46	47437,48	42510,23
20	41099,235	15532,525	46035,97	27410,78499

Liite 2. Web SQL Databasen Crud-testin ajat millisekunteina

100k test	Create	Read	Update	Delete
1	32638,325	521,175	591,6199	221,345
2	31399,605	582,0399	628,65	225,15
3	9087,39	589,68499	603,6399	231,005
4	26457,35	649,47499	600,845	224,08
5	11366,99	647,345	621,53499	229,83499
6	39491,645	645,15	618,985	227,199
7	38299,96	636,485	645,52	231,35
8	38127,03	568,125	647,63	241,6
9	8580,415	638,15	614,49	234,5599
10	29285,995	661,125	592,22499	231,39
11	39611,625	560,499	621,78	229,92499
12	9532,15	639,41499	636,985	219,9799
13	39709	610,22499	596,985	201,5499
14	41112,17	584,91	603,19	223,3399
15	41804,29	605,6199	613,53499	224,97499
16	9056,005	519,03	608,7299	225,085
17	9241,94	617,665	589,34499	192,8699
18	21191,21	596,51	598,01	223,11499
19	19162,19	614,9799	599,8899	218,305
20	8370,415	623,455	603,18499	222,07