Hannes Ranta

SOVELLUKSEN TIETOKANNAN UUDELLEENSUUNNITTELU

Tietojenkäsittelyn koulutusohjelma

2016

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

SOVELLUKSEN TIETOKANNAN UUDELLEENSUUNNITTELU

Ranta, Hannes
Satakunnan ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Marraskuu 2016
Ohjaaja: Nieminen, Hans
Sivumäärä: 33
Liitteitä: 3

Asiasanat: VBScript, JSON, SQL, relaatiotietokannat

Opinnäytetyön aiheena oli tietokannan uudelleensuunnittelu ja kahden järjestelmän välisen viestinnän mahdollistaminen JSONilla. Projektin tilasi Hakosalo Software Oy.

Opinnäytetyön oli tarkoitus olla osa laajempaa koiranäyttelyprojektia ja käytännönläheisempi kuin mitä se nyt on. Tämän projektin epävarma tulevaisuus kuitenkin aiheutti sen, että opinnäytteestä tehtiin enemmän teoriaa painottava ja tulevaisuusvarmempi.

Opinnäytetyö keskittyy voimakkaasti tietokannan suunnittelun teoriaan ja käy läpi sen eri vaiheet ER-kaavioista fyysiseen suunnitteluun. Käytännön puolella käydään läpi suunnitteluprojektin eteneminen vaiheittain loppuun asti. Projekti oli hyvin haasteellinen ja opettavainen.

REDESIGN OF APPLICATION'S DATABASE

Ranta, Hannes
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Business Information Systems
November 2016
Supervisor: Nieminen, Hans
Number of pages: 33
Appendices: 3

Keywords: VBScript, JSON, SQL, relational databases

The purpose of this thesis was to redesign an application's database and to enable communication between two systems using JSON. The project was requested by Hakosalo Software Oy.

The thesis was meant to be a part of a larger conformation show project and was supposed to be more practical than what it ended up being. The uncertain future of the project however caused it to focus more on theory and being futureproof.

This thesis focuses strongly on database design theory and walks through it's different phases from ER diagrams to physical design. In the practical portion this thesis goes through the steps it took to finish the project. The project was very challenging and educational.

SISÄLLYS

1 JOHDANTO

Opinnäytetyön tarkoituksena on suunnitella uudelleen jo olemassa olevan järjestelmän tietokanta sekä uudistaa järjestelmään tuotavien tietojen tuonti helpommin käytettävään malliin. Opinnäytetyön asiakasyrityksenä toimii Hakosalo Software Oy, joka tuottaa muun muassa web-pohjaisia CRM-, CMS- ja ajanhallintaohjelmistoja.

Hakosalo Software Oy on hyvin läheinen tuttavuus asiakasyrityksenä minulle, koska suoritin siellä opintoihini kuuluvan 5 kuukauden harjoittelujakson ja tämän jälkeen tein noin puoli vuotta koulun ohella harjoittelijana töitä kyseiselle yritykselle. Juuri tämän jakson loppupuolella tein opinnäytetyösopimuksen tietokannan uusimisesta, joka oli tämän projektin osa, jonka parissa olin työskennellyt.

Projektissa on luotu koiranäyttelyille ilmoittautumis-, tulos- ja hallintajärjestelmät, joita tuli kehittää eteenpäin ja pyrkiä myös yhdistämään. Lopuksi uusi tietokanta tulisi toimimaan yhdistetyn järjestelmän ainoana kantana. Kuitenkin tätä ennen järjestelmässä olisi tarkoitus käyttää kahta tietokantaa rinnakkain, joka tuottaa ongelmia johtuen kantojen sijainnista täysin eri järjestelmissä. Tähän ongelmaan on tarkoitus tehdä väliaikaiseksi ratkaisuksi JSON-rajapinta, jonka avulla saadaan tietojensiirtoa helpotettua ennen lopullisen järjestelmän valmistumista.

Työssä tullaan käsittelemään tietokannan suunnittelu painottaen teoriaa ja tämän lisäksi käydään läpi JSON-rajapinnan luonti. Turvallisuuden ja oikeuksien turvaamiseksi en käytä esimerkeissä alkuperäisiä koodeja tai tietokantoja, vaan näiden mukaisversioita. Sisältö on kuitenkin sovellettavissa vastaaviin ympäristöihin, joten informaatiota ei menetetä tämän metodin vuoksi.

Opinnäytetyön tekemisen aikana projektin tilanne on myös kokenut joitakin muutoksia ja on tämän kirjoittamishetkellä hyvin epävakaalla pohjalla. Tästä johtuen opinnäytetyö painottuu enemmän teoreettiseen osuuteen.

2 KÄYTETTÄVÄT TEKNIIKAT

2.1 Active Server Pages eli Classic ASP

Classic ASP julkaistiin vuonna 1996 ja se oli Microsoftin ensimmäinen palvelinpuolen skriptimoottori, joka tarjosi mahdollisuuden verkkosivujen dynaamiseen luomiseen ja tätä kautta antoi mahdollisuuden luoda web-pohjaisia ohjelmistoja pelkkien staattisten sivujen sijaan. Käytännössä tämä tarkoittaa, että sivuston sisältö voi muuttua dynaamisesti käytön mukaan esimerkiksi silloin, kun käyttäjä kirjautuu sisään sivustolle. Nimitys Classic ASP ei ole tekniikan alkuperäinen nimi, vaan sitä käytetään nykyään pääasiassa estämään sekaannusta tämän seuraajaan, ASP.NETtiin. Classic ASPin kehittäminen on loppunut vuonna 2000 versionumeroon 3. (Technopedia 2016; Microsoftin tuen www-sivut 2016; Computer Hope 2016.)

Hyvä esimerkki Classic ASPin käytöstä on tietokantayhteys. Alla olevassa kuvassa on esitetty yksinkertainen tapaus, jossa tietokantaan on otettu yhteyttä ja sieltä on haettu dataa. Aluksi tietokantayhteys avataan, tämän jälkeen suoritetaan SQL-kysely, jonka tulos asetetaan muuttujaan. Lopuksi yhteys vielä suljetaan. Tästä kyselystä syntyy tuloksena kaikkien tähän asti käytyjen kisojen ensimmäisten kehien tiedot. (Mar 2016.)

```
Dim sConn
Dim rs
Dim sSql

set sConn = Server.CreateObject("ADODB.Connection")

sConn.ConnectionString = strConn
sConn.Open

sSql = "SELECT * FROM ROUND WHERE Round_number = 1;"
Set rs = sConn.execute(sSql)

sConn.Close
set sConn = Nothing
```

Kuva 1. Classic ASPin mukainen tietokantayhteyden hallinta

2.2 VBScript eli Visual Basic Scripting Edition

VBScript on Microsoftin vuonna 1996 julkaisema skriptauskieli, jota on käytetty eritoten Classic ASPin kanssa tuomaan sivustoille palvelinpään skriptausta ja toiminnallisuutta. VBScriptiä voi myös käyttää Windows Scripting Hostin kanssa Windows-ympäristöjen automatisointiin. Se on ottanut huomattavasti vaikutteita ja toimintoja nimikkokielestään, Visual Basicista. (Smiley 2015; Tutorialspoint 2016.)

Kuten Classic ASPin kohdalla, myös VBScriptin kehitys on loppunut huomattava aika sitten, mutta Microsoft korjaa vielä niitä tietoturva-aukkoja, joita löydetään. Viimeisen version numeroksi on jäänyt 5.8. Käytännössä C#:sta on tullut käytetyin kieli ASP.NET-ympäristössä ylläpitäen siis samaa asemaa kuin VBScript ylläpiti Classic ASPissa. (Tutorialspoint 2016.)

Jatketaan edellä luotua esimerkkiä. Edellisessä vaiheessa tehtiin kysely, josta saatiin kaikki tähän asti käytyjen kisojen ensimmäisten kehien tiedot. Alla on kuvattu, miten nämä tiedot voitaisiin näyttää nettisivulla. Siinä HTMLän joukkoon upotettu VBScript-koodi käy läpi jokaisen yksittäisen kisan ja luo uuden rivin jokaiselle päivälle. Lopuksi yhteys suljetaan.

```
<h2>Kisojen ensimmäisten kehien päivämäärät:</h2>
<ul>
   <%
      do until rs.eof
         Response.Write ("<li>" & rs("Comp_date") & "</li>")
         rs.movenext
      loop
      rs.close
   %>
</ul>
```

Kuva 2. HTMLän joukkoon upotettua VBScriptiä

2.3 SQL

SQL-kielen nimi tulee sanoista Structured Query Language, rakenteellinen kysely-kieli. Kieli on IBM Researchin 1970-luvun alkupuolella kehittämä ja ensimmäinen järjestelmä, missä sitä käytettiin laajasti, oli System R. (Date, Kannan & Swamynathan 2003, 69.)

SQL-kieltä käytetään lähes kaikissa relaatiotietokantajärjestelmissä kyselykielenä, jonka avulla tietoja haetaan järjestelmästä. Sen avulla voidaan myös luoda, muokata ja poistaa tietokantoja ja näiden sisältöä. Täten se toimii hyvin tärkeässä roolissa lähes kaikissa voimakkaasti informaation säilömiseen ja käsittelemiseen liittyvissä tietojenkäsittelyprojekteissa. Alla on kuvattu joitakin esimerkki SQL-lauseita.

```
CREATE DATABASE EsimerkkiTietokanta;

CREATE TABLE HENKILO (
Etunimi varchar(30),
Sukunimi varchar(30)
);

INSERT INTO HENKILO (Etunimi, Sukunimi)
VALUES ('Eetu', 'Esimerkki');

ALTER TABLE HENKILO
ADD Ika int
```

Kuva 3. Muutamia SQL-lauseita

2.4 SQL SERVER 2008

Microsoft SQL Server 2008 on Microsoftin kehittämä relaatiotietokantahallintajärjestelmä, jossa käsittelykielenä käytetään SQL-kieltä. SQL Server 2008 on julkaistu vuonna 2008 ja Microsoft on luvannut tukea sitä aina vuoteen 2019 asti. (Microsoft 2014.)

Eri tietokantahallintajärjestelmien välillä on joitakin eroja. Erot voivat olla monia eri asioita koskevia aina tuetuista kielistä SQL-lausekkeisiin. Matalammalla tasolla voidaan sanoa, että eri järjestelmät toteuttavat eri tavoin tai eri osin relaatiomallia. Alla

olevassa kuvassa on tehty vertailua kolmen suositun järjestelmän välillä, mukaan lukien SQL Server. (Lee 2013.)

Feature	Oracle	MySQL	SQL Server
Interface	GUI, SQL	SQL	GUI, SQL, Various
Language support	Many, including C, C#, C++, Java, Ruby, and Objective C	Many, including C, C#, C++, D, Java, Ruby, and Objective C	Java, Ruby, Python, VB, .Net, and PHP
Operating System	Windows, Linux, Solaris, HP-UX, OS X, z/OS, AIX	Windows, Linux, OS X, FreeBSD, Solaris	Windows
Licensing	Proprietary	Open source	Proprietary

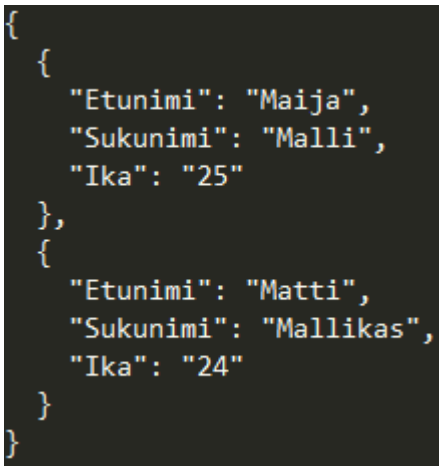
Kuva 4. Vertailua kolmen tietokantahallintajärjestelmän välillä (Lee 2013.)

Sivumainintana on pakko tuoda esille, että Microsoft on tuomassa helpotusta JSONin käyttämiseen tämän projektin tulevaisuudessa ja muihin vastaaviin projekteihin: SQL Server tulee versiosta lähtien 2016 tarjoamaan JSON-tuen. Tämä tarkoittaa sitä, ettei tässä projektissa tehtyä työtä JSON-tulkin käyttöönottamiseksi enää tarvitse tehdä, mikä itsessään helpottaa ja nopeuttaa testaamista. Vaikka kyseessä ei olekaan natiivi JSON-tyypin tuki, niin tarjoaa SQL Server 2016 suunnitelmien mukaan tuen datan tuontia ja vientiä varten JSON-muodossa. (Popovic 2015.)

2.5 JSON

ECMA Internationalin, joka on standardeja tuottava organisaatio, mukaan JSON on ”Tekstiformaatti, joka helpottaa rakenteellisen datan vaihtoa eri ohjelmointikielten välillä.” Se on ottanut voimakkaasti vaikutteita JavaScriptin objektien vakioarvoilmaisuista eli pilkuilla erotellusta listasta avain-arvopareja. (ECMA International 2013, ii.)

Alla on vielä kerran jatkettu esimerkkiä, joka aloitettiin alussa. Tässä kuvassa tuodaan HENKILO-tauluun sopivaa dataa eli kaksi uutta henkilöä. JSONia voidaan käyttää astiana siirrettävälle datalle esimerkiksi silloin, kun halutaan viikoittain uusi ruokalista. Tällöin voidaan tiedot viedä JSON-tiedostolla kerran viikossa sen sijaan, että sivustoa itsessään päivitettäisiin tai tietokantaan otettaisiin jokainen kerta uudestaan yhteyttä.



Kuva 5. Esimerkki JSONista

3 TIETOKANTASUUNNITTELUN TEORIA

Tietokantasuunnittelun perimmäinen tarkoitus on tuottaa hyvä tietokanta. Käytännössä tämä tarkoittaa kantaa, jonka käyttäminen sovellusohjelmalla on helppoa, josta tietojen hakeminen on nopeaa ja joka on tuotteen eliniän kattava, joko hyvän mallinsa tai muutosjoustavuuden vuoksi.

Tietokannan suunnittelu voidaan karkeasti jakaa kolmeen vaiheeseen:

- tietokannan looginen suunnittelu
- tietokannan loogisen rakenteen toteuttaminen
- sovelluksen kehittäminen.

Ensimmäinen vaihe sisältää itse kannan suunnittelun: taulujen ja näiden sarakkeiden määrittäminen, perus- ja viiteavainten määrittäminen ja muut tarvittavat toiminnot. Toinen vaihe taas sisältää kannan fyysisen luomisen jollakin tietokantaohjelmistolla.

Kolmannessa vaiheessa puolestaan tehdään sovellus, jota loppukäyttäjä tulee käyttämään. (Hernandez 2000, sivu xxx.)

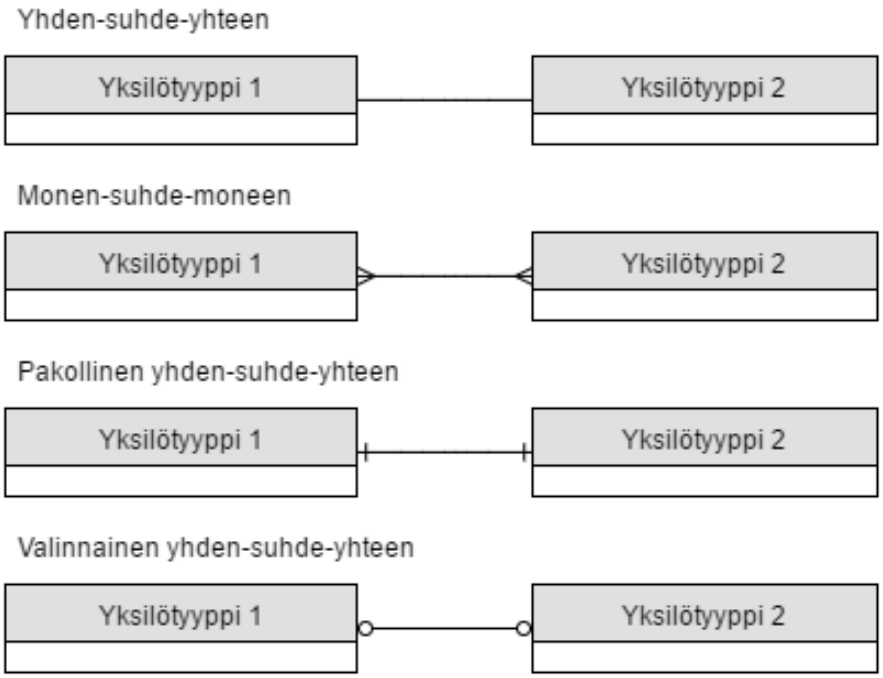
Tämä luku erityisesti ja opinnäytetyö yleisesti käsittelee näistä vaiheista voimakkaimmin kahta ensimmäistä. Luku itsessään on jaettu alalukuihin, joista kukin käsittelee yhtä tietokantasuunnittelun vaihetta. Ja vaikka luvut ovatkin tietyssä järjestyksessä, on hyvä tiedostaa, että usein käytännön työssä lukujen välillä saatetaan liikkua hyvinkin vapaasti tarpeen vaatiessa, suunnittelutyön usein ollessa hyvin iteratiivista lineaarisen sijasta.

3.1 Käsiteanalyysi eli käsitteellinen suunnittelu

Käsiteanalyysissä on tarkoituksena luoda tulevasta tietokannasta karkea malli, runko, jonka päälle myöhemmät vaiheet rakentuvat. Tämä käsitemalli kuvaa esimerkiksi nimitasolla tietokannan rakennetta, mutta ei huolehdi vielä tietotyypeistä. Tämän vaiheen korkea abstraktion taso on omiaan tekemään yhteistyöstä asiakkaan kanssa suunnittelun parissa huomattavasti helpompaa, etenkin kun asiakas ei aina ole tietojenkäsittelyalan ammattilainen.

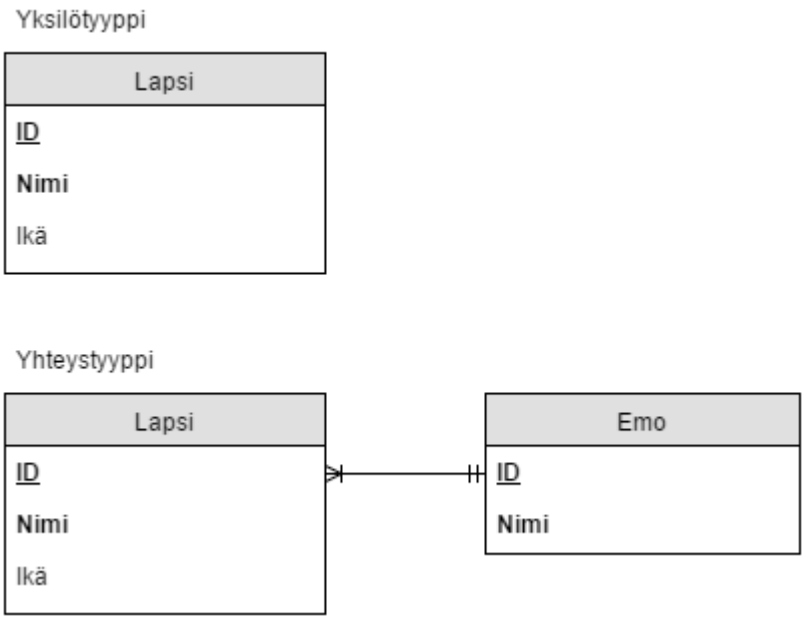
Tällaista mallia kutsutaan nimellä ER-malli ja se tulee sanoista Entity-Relationship Model. Erilaisia ER-malleja kuvataan kuvaustekniikoilla eli notaatioilla. Tässä työssä käytetty notaatio on niin kutsuttu Crow’s foot notation eli harakanvarvasnotaatio. Sillä kuvataan yksilötyyppejä eli entiteettejä, näiden yksilötyyppien yhteystyyppejä eli suhteita ja näiden yhteystyyppien pakollisuutta. Toinen tapa mallintaa rakennetta on luokkakaavio, mutta en käsittele sitä tässä työssä. (Hovi, Huotari, Lahdenmäki 2005, 33; Nieminen 2009, 32.)

Seuraavalla sivulla olevassa kuvassa on esitetty joitakin yleisimpiä suhteita. Näiden suhteiden yhdistelmällä päästään hyvin pitkälle tietokantasuunnittelun tässä vaiheessa. Yhteystyyppejä voidaan myös nimetä suhteen selkeyttämiseksi.



Kuva 6. Osittainen katsaus harakanvarvasnotaation merkitsemistapoihin

Seuraavalla sivulla olevassa kuvassa esitetään yksilötyyppi nimeltä Lapsi, jolle on määritelty ominaisuudet eli attribuutit ID, Nimi ja Ikä. Yksilötyyppien ominaisuuksille on hyvä tässä vaiheessa jo merkitä pakollisuus lihavoimalla ominaisuuden nimi. Tämän lisäksi on suositeltavaa merkitä alleviivaamalla ne ominaisuudet, jotka toimivat yksilötyypin perusavaimena. Sellaisena voi toimia attribuutti tai attribuuttien joukko, joka tuottaa uniikin arvon. Suomalaiselle tällainen attribuutti olisi henkilötunnus ja seuraavassa esimerkissä tällaisena toimii ID.



Kuva 7. Lapsi-yksilötyyppi ja Lapsen ja Emon välisen yhteyden kuvaus

Edellä olevassa kuvassa yksilötyypin alapuolella on Lapsi- ja Emo-yksilötyypit, joiden välillä on yhteystyyppi. Tämä suhde tarkoittaa, että Lapsi-yksilötyyppiin liittyy tasan yksi Emo-yksilötyyppi ja vuorostaan Emo-yksilötyyppiin voi liittyä yksi tai useampia Lapsi-yksilötyyppiä. Molemmilla yksilötyypeillä on ID ja Nimi pakollisina ominaisuuksina eli myöhemmin näillä tulee olla aina jokin arvo, mutta Lapsen Ikä-ominaisuus on jätetty vapaaehtoiseksi.

Yksilötyyppien ja yhteystyyppien avulla saadaan suunniteltua tietokanta karkealla tasolla. Tärkeää suunnittelussa on muistaa, ettei tietokannan tulisi sisältää ylimääräistä tietoa, mutta ei myöskään toisaalta liian vähän: puutteellinen tietokanta ei voi ikinä toteuttaa toimintoaan, mutta ylitäysi tietokanta voi, vaikka ehkä tuottaen ongelmia ohjelmistokehitykselle tai ylläpidolle. Kanta tulisi myös yrittää suunnitella alusta asti mahdollisimman vähän tietoa toistavaksi.

3.2 Tarveanalyysi ja looginen suunnittelu

Tarveanalyysin tarkoituksena on täsmentää, tarkentaa ja täydentää edellisessä vaiheessa luotua käsitemallia. Tarkoituksena on käydä läpi erilaisia tietotarpeita ja niiden pohjalta lisätä käsitemalliin kaikille yksilötyypeille ominaisuudet ja niiden pakollisuudet. Tämän jälkeen täydennetään käsitemallia lisäämällä myös havaitut puut-

tuvat yhteystyypit sekä yksilötyypit. Tarkoituksena on, että tämän vaiheen jälkeen tietokannan looginen rakenne olisi suhteellisen lähellä lopullista ja myöhemmät vaiheet ovatkin enemmän tähänastisen työn hiomista käyttövalmiiksi. Siksi tähän työvaiheeseen olisi tärkeää panostaa voimakkaasti. (Hovi ym. 2005, 80)

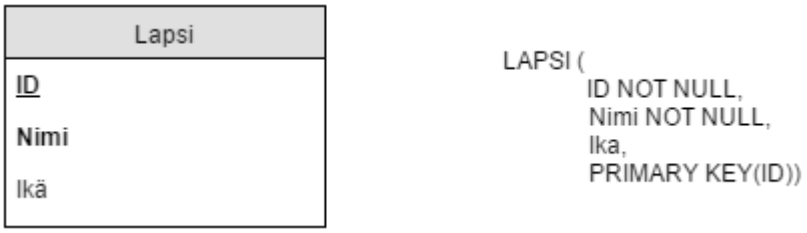
Tarveanalyysin jälkeen siirrytään loogiseen suunnitteluun. Tämän tarkoituksena on tuottaa tietokantakaava, josta nähdään, millaisilla rakenteilla käsitemallin tiedot on tarkoitus tallettaa tietokantaan. Tässä suunnittelun vaiheessa aletaan nähdä, miten tietokannan rakenne muodostuu. Käytännössä tämä suunnittelutyö on kolmivaiheinen:

- relaatioiden muodostus
- viiteavainten muodostus
- arvojoukkojen muodostus.

(Nieminen 2009, 43.)

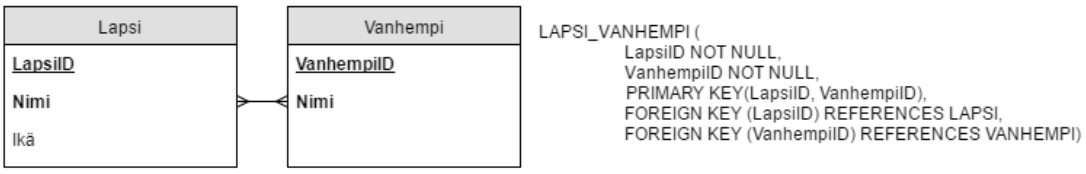
3.2.1 Relaatioiden muodostus

Ensimmäisessä vaiheessa muodostetaan relaatiot jokaiselle käsitemallin sisältämälle yksilötyypille. Käytännössä tämä tarkoittaa mekaanista prosessia, jossa yksilötyypeistä, monen-suhde-moneen -yhteyksistä, ominaisuuksia sisältävistä yhteyksistä, monimutkaisista yhteyksistä ja moniarvoisista ominaisuuksista muodostetaan relaatiot. Tämän prosessin jälkeen on saatu luotua kaikki relaatiot pois lukien ne jotka syntyvät normalisoinnin myötä. Alapuolella olevassa kuvassa näytetään, miten tämä yksinkertaisimmillaan tarkoittaa yksilötyypin yksiarvoisten ominaisuuksien muuttamista attribuuteiksi ja perusavaimen määrittämistä. (Nieminen 2009, 45.)



Kuva 8. Lapsi-yksilötyyppi ja siitä muodostettu relaatio. Tietojen pakollisuus on osoitettu NOT NULL -merkinnällä sekä ääkkösistä on luovuttu, kuten ohjelmoinnissa yleensäkin.

Jokainen monen-suhde-moneen -yhteys puretaan omaksi relaatiokseen. Tällaisessa relaatiossa on molempien osapuolina olevien relaatioiden perusavaimet viiteavaimina, jonka lisäksi näiden yhdistelmä muodostaa yhdistelmärelaation perusavaimen. Alla olevassa kuvassa esitellään jälleen yksilötyyppi Lapsi, mutta Emo on korvattu Vanhemmalla ja Lapsella on tämän kanssa monen-suhde-moneen -yhteys. Tässä kuvassa näkee, miten tämä yhteys muutettaisiin omaksi relaatiokseen. (Nieminen 2009, 46.)



Kuva 9. Monen-suhde-moneen -yhteyden muuttaminen relaatioksi

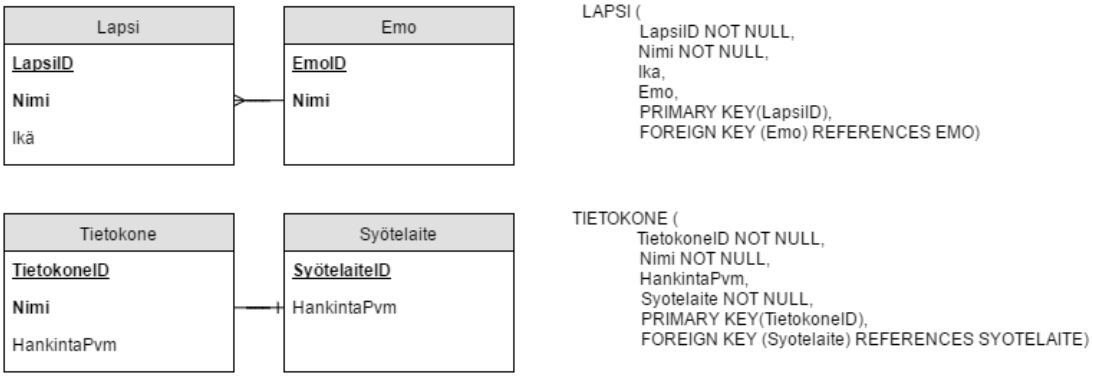
3.2.2 Viiteavainten muodostus

Toisessa vaiheessa muodostetaan viiteavaimet relaatioihin yksilötyyppien välisten yhteyksien perusteella. Viiteavaimen muodostavia yhteyksiä ovat yhden-suhde-moneen -yhteys, yhden-suhde-yhteen -yhteys sekä vaihtoehtoiset yhteydet. Käyn tässä läpi vain yhden-suhde-moneen ja yhden-suhde-yhteen -yhteydet tilan säästämiseksi. (Nieminen 2009, 54.)

Yhden-suhde-moneen -yhteydet muodostavat viiteavaimen siihen relaatioon, mikä sijaitsee yhteyden ”monen” päässä. Tämän lisäksi tämä viiteavain merkitään NOT NULL -merkinnällä, kun yhteys on merkitty pakolliseksi ”yhden” päässä. Seuraavalla sivulla olevassa kuvassa näytetään esimerkki tällaisesta suhteesta käyttäen Lapsen ja Emon välistä suhdetta. (Nieminen 2009, 54.)

Yhden-suhde-yhteen -yhteyksissä viiteavain muodostetaan jompaankumpaan relaatioon. Hovin mukaan todellisia yhden-suhde-yhteen -yhteyksiä ilmenee harvoin ja ne jakavat silloinkin usein suurimman osan tiedoista toistensa kanssa, jolloin kahden relaation yhdistäminen yhdeksi on nykyisten suositusten mukaan turvallinen valinta. Valittava relaatio on yleensä se, johon viiteavaimen muodostaminen tuottaa vähem-

män NULL-arvoja. Viiteavain tulee merkitä NOT NULL -merkinnällä, jos yhteys on merkitty pakolliseksi vastakkaisessa päässä. Alla olevassa kuvassa on nähtävissä esimerkki tällaisesta suhteesta Tietokoneen ja Syötelaitteen välisenä tapahtumana. (Hovi ym. 2005, 80; Nieminen 2009, 56.)



Kuva 10. Yhden-suhde-moneen -yhteyden ja yhden-suhde-yhteen -yhteyden muuntaminen relaatioksi

3.2.3 Arvojoukkojen muodostus

Tässä vaiheessa relaatioiden attribuuteille määritellään arvojoukot, jotka ovat SQL Serverin tapauksessa valmiiksi määriteltyjä. Käytännössä tämä tarkoittaa, että käydään jokainen relaatio yksi kerrallaan läpi ja tämän jokaiselle attribuutille määritetään tietotyyppi kuten INT, DATE tai VARCHAR. Arvojoukkoja voi myös itse määritellä, mutta kaikki tietokantahallintajärjestelmät eivät välttämättä tue niitä. On hyvin tärkeää, että jokaiselle attribuutille määritellään sopiva tietotyyppi ja että viiteavaimelle ja perusavaimelle johon se viittaa, määritellään sama tietotyyppi. (Nieminen 2009, 62.)

3.3 Normalisointi

Normalisoinnin tarkoituksena on tuottaa relaatioille parempi muoto tiettyjä hyväksi havaittuja sääntöjä käyttäen. Sen voidaankin ajatella olevan tietokantasuunnittelun tarkastaja. Normalisoitua tietokantaa on nopea päivittää, tietojen toistaminen on minimoitu ja se on muutosjoustava. Normalisointi perustuu niin kutsuttuihin normaali-

muotoihin, joista jokainen tarvitsee pohjakseen kaikki aiemmat. Erilaisia normaalimuotoja ovat:

- 1. normaalimuoto eli 1NF
- 2. normaalimuoto eli 2NF
- 3. normaalimuoto eli 3NF
- Boyce-Codd -normaalimuoto eli BCNF
- 4. normaalimuoto eli 4NF
- 5. normaalimuoto eli 5NF

Näiden lisäksi relaatioita voidaan vielä denormalisoida, joka tarkoittaa normalisoinnin tahallista ja hallittua poistamista tehokkuussyistä. (Hovi ym. 2005, 86; Nieminen 2009, 65.)

Vaikka erilaisia normaalimuotoja on kuusi erilaista, niin yleisesti käytännössä niitä käytetään vain 3. normaalimuotoon asti. Tämä tarkoittaa, että normalisoimaton relaatio normalisoidaan 1NF mukaiseksi, sen jälkeen 2NF mukaiseksi ja loppujen lopuksi 3NF mukaiseksi. Tämän jälkeen joissakin erityistapauksissa, kuten kyselypainotteisissa tietovarastotietokannoissa relaatioita vielä denormalisoidaan kyselyiden nopeuttamiseksi.

Tässä osiossa käytetään myös termiä funktionaalinen riippuvuus. Se tarkoittaa sitä, että attribuutti X määrää attribuutin Y, kun jokaiseen X:n arvoon liittyy täsmälleen yksi Y:n arvo. Käytännössä tämä voidaan esittää relaatiolla, joka sisältää attribuutit: ID ja Nimi. ID määrää attribuutin Nimi, kun jokaisen ID:n arvoon liittyy täsmälleen yksi Nimen arvo. Tämä siis toteutuu. Toisinpäin ajatellen voidaan todeta, että yhdellä nimellä voidaan saada tulokseksi monta ID:tä, jolloin kyseessä ei ole funktionaalinen riippuvuus. Tässä esimerkissä ID:tä kutsutaan määrääjäksi ja Nimeä kutsutaan määrättäväksi. (Hovi ym. 2005, 90; Nieminen 2009, 68.)

3.3.1 Normalisoimaton muoto

Relaatio on normalisoimaton silloin, kun se sisältää attribuutin tai attribuuttien joukon, jolla on useita arvoja jokaista yksittäistä relaation perusavainta kohti. Tällaista

muotoa kutsutaan nimellä UNF. Alla olevassa kuvassa on kuvattu normalisoimaton relaatio. (Nieminen 2009, 66.)

TietokoneID	Nimi	Syotelaite1	Syotelaite2	Syotelaite3	LaiteLkm
1	Toimisto1	Hiiri	Nappaimisto		2
2	Toimisto2	Hiiri	Nappaimisto	Piirtopoyta, mikrofoni	3
3	Aula	Hiiri	Nappaimisto		2
4	Varasto	Hiiri	Nappaimisto	Viivakoodinlukija	3

Kuva 11. Normalisoimaton relaatio Tietokone

3.3.2 Ensimmäinen normaalimuoto

Ensimmäinen normaalimuoto eli 1NF toteutuu silloin, kun relaatiossa ei ole yhtään moniarvoista attribuuttia tai toistuvia ryhmiä eikä se sisällä johdettua dataa. Edellä esitetystä kuvasta on mahdollista löytää kaikkiin 1NF:n sääntöihin kohdistuvia rikkeitä. Näistä pahin on LaiteLkm:n johdettu data: se ei ole laskettu oikein johtuen Syotelaite3:n moniarvoisuudesta. (Hovi ym. 2005, 87; Nieminen 2009, 68.)

Tämä relaatio saadaan muutettua 1NF:n mukaiseksi kolmella vaiheella. Ensiksi moniarvoiset attribuutit pitää purkaa joko erillisiksi attribuuteiksi tai sitten niistä pitää tehdä oma relaationsa. Tämän jälkeen relaatiosta poistetaan johdetut attribuutit. Ja viimeisenä toistoryhmille luodaan oma relaatio, johon ne siirretään. Alla olevasta kuvasta näkee, millainen lopputulos tästä voi syntyä. (Hovi ym. 2005, 87; Nieminen 2009, 68.)

TietokoneID	Nimi
1	Toimisto1
2	Toimisto2
3	Aula
4	Varasto

SyotelaiteID	Nimi
1	Hiiri
2	Nappaimisto
3	Piirtopoyta
4	Mikrofoni
5	Viivakoodinlukija

TietokoneID	SyotelaiteID
1	1
1	2
2	1
...	...

Kuva 12. Tietokone-, Syotelaite- ja TietokoneSyotelaite-relaatiot toteuttaen 1NF:n

3.3.3 Toinen normaalimuoto

Toinen normaalimuoto eli 2NF toteutuu silloin, kun relaatio on 1NF:ssa ja jokainen perusavaimeen sisältymätön attribuutti on kokonaisriippuvainen perusavaimesta eli se on funktionaalisesti riippuvainen perusavaimen jokaisesta attribuutista eikä vain osasta näitä. Tämä ei ole useinkaan nykyään ongelma, koska hyvin monesti taulun perusavaimena toimii surrogaattiavain eli ohjelmallisesti päivittyvä juokseva numero. Tämä voi tietysti aiheuttaa liitostarpeita kyselyissä kuten edellisestä kuvasta nähdään. (Hovi ym. 2005, 63; Nieminen 2009, 70.)

Relaatio voidaan muuttaa 2NF:n mukaiseksi, jos se on 1NF:n mukainen ja sisältää moniattribuuttisen perusavaimen. Ensiksi pitää löytää riippuvuudet, joissa määrääjä on perusavaimen osa ja määrättävä attribuutti on perusavaimeen kuulumaton. Tämän jälkeen luodaan uusi relaatio, jonka attribuuteiksi tulevat kaikki edellä löydetyt määrääjät ja määrättävät attribuutit. Tämän relaation perusavaimeksi tulevat löydetyt määrääjäattribuutit. Lopuksi alkuperäisestä relaatiosta poistetaan määrättävät attribuutit. Alla on Hovin esimerkki 2NF:n toteuttamisesta. (Hovi ym. 2005, 63; Nieminen 2009, 70.)

Toimittajatun	Osanro	Pvm	Toim_nimi	Osanimi	Määrä
1202	707b	4.6.2003	Varaosa Oy	Tulppa 12	22
1202	708b	3.4.2003	Varaosa Oy	Tulppa 13	10
1275	707b	15.5.2003	A-osat Oy	Tulppa 12	5
...					

Toimittajatun	Toim_nimi
1202	Varaosa Oy
1275	A-osat Oy
...	

Osanro	Osanimi
707b	Tulppa 12
708b	Tulppa 13
...	

Toimittajatun	Osanro	Pvm	Määrä
1202	707b	4.6.2003	22
1202	708b	3.4.2003	10
1275	707b	15.5.2003	5
...			

Kuva 13. Esimerkki 2NF:n toteutuksesta. Yllä alkuperäinen relaatio, alla 2NF:n muodon mukainen toteutus. (Hovi ym. 2005, 91.)

3.3.4 Kolmas normaalimuoto

Kolmas normaalimuoto eli 3NF toteutuu silloin, kun relaatio on 2NF:ssa ja jokainen perusavaimeen kuulumaton attribuutti on funktionaalisesti riippuvainen pelkästään perusavaimesta eikä mistään muusta osasta relaatiota. Kolmannen normaalimuodon tarkoitus onkin varmistaa, ettei mikään perusavaimen ulkopuolinen attribuutti ole riippuvainen muusta kuin perusavaimesta. (Hovi ym. 2005, 93; Nieminen 2009, 72.)

Jotta relaatio voidaan muuttaa 3NF:n mukaiseksi, pitää siitä ensiksi etsiä kaikki kokonaisriippuvuudet, joiden määrääjä ei ole perusavain. Jokaiselle löydetylle kokonaisriippuvuudelle muodostetaan uusi relaatio, jonka attribuutit ovat kokonaisriippuvuuden määrääjä ja määrätty. Määrääjä toimii näissä relaatioissa perusavaimena. Lopuksi tulee alkuperäisestä relaatiosta poistaa määrätty. Alla on kuvattu esimerkki, miten 3NF toteutetaan. (Hovi ym. 2005, 93; Nieminen 2009, 72.)

TUOTE

TuoteId	Nimi	Valmistaja	ValmistajaMaa
1001	Walkman	Sony	Japani
1002	Kamera	Leica	Saksa
1003	DVD	Sony	Japani

TUOTE

TuoteId	Nimi	Valmistaja
1001	Walkman	Sony
1002	Kamera	Leica
1003	DVD	Sony

VALMISTAJA

Valmistaja	ValmistajaMaa
Sony	Japani
Leica	Saksa

Kuva 14. Alemmat kaksi relaatiota ovat lopputulos ylemmän relaation muuttamisesta 3NF:iin. (Nieminen 2009, 73.)

3.3.5 Denormalisointi

3NF voi aiheuttaa joitakin hieman erikoisia tilanteita, kuten normalisoida Asiakkaat-relaatiosta postitoimipaikan omaksi relaatiokseen, kun käsitellään asiakkaan osoitetietoja. Tällöin tulee miettiä, voidaanko tätä ratkaisua hyödyntää vai tulisiko tämä relaatio denormalisoida kyselyiden helpottamiseksi. Denormalisointi on tietokantasuunnittelun vaihe, jossa relaatioita jotka ovat 3NF:ssa rikotaan 2NF:iin. Tämä teh-

dään puhtaasti tehokkuussyistä, sillä tätä kautta saadaan vähennettyä kyselyissä tapahtuvia liitoksia. Tästä syystä denormalisointi on yleinen ratkaisu tietovarastokannoissa, joihin kohdistuu useita kyselyitä. Jos tieto on usein päivittyvää, ei denormalisointi ole todennäköisesti sille oikea ratkaisu. (Hovi ym. 2005, 94; Nieminen 2009, 77.)

Denormalisoinnissa on muutamia ongelmia. Ensimmäisenä näistä on, että jos tietokantaa ryhdytään denormalisoimaan, kuinka pitkälle sitä tulisi tehdä? Käytännöksi on muodostunut normalisointiin voimakkaasti sidoksissa oleva tapa, jolloin tietokanta joka on 3NF:ssa voidaan joiltakin osiltaan denormalisoida 2NF:n mukaiseen muotoon. Silloin tietoja, jotka olisivat normaalisti viiteavaimen päässä, siirretään viitattavaan riviin. Tätä voi ajatella funktionaalisesti etukäteen tehtynä liitoskyselynä. Toisena ongelmana on tietojen päivittäminen ja sen monimutkaistuminen, mutta näiden lisäksi joistakin kyselyistäkin voi tulla hankalampia suorittaa. Ratkaisuna tähän ongelmaan voisi olla laukaisimien käyttö tietokannassa, jotka automaattisesti osaisivat ylläpitää tiedon eheyttä monessa paikassa. Lopputulos on, että denormalisointi voi olla tehokas ratkaisu, mutta sitä tulee käyttää varovaisesti. (Date ym. 2003, 329; Hovi ym. 2005, 95; Nieminen 2009, 77.)

3.4 Eheyssäännöt ja tietokannan fyysinen toteutus

Tämä luku sisältää tietokannan eheyssääntöjen määrittelyn sekä myös nopean katsauksen indeksointiin. Tässä vaiheessa on tietokanta lähestulkoon jo valmis. Fyysinen suunnittelu tässä tilanteessa tarkoittaisi lähinnä luomislauseiden toteuttamista jollekin tietokannanhallintajärjestelmälle sovellettuna.

3.4.1 Eheyssäännöt

Tässä vaiheessa viimeistään tulee merkitä kaikille pakollisille ominaisuuksille NOT NULL -määritys, jos niitä ei ole aiemmin merkitty relaatioihin. Tämän jälkeen viiteavaimille tulee merkitä poisto- ja päivityssäännöt. Erilaisia sääntöjä ovat, ON DELETE/UPDATE:

- RESTRICT

- CASCADE
- SET DEFAULT
- SET NULL.

Restrictillä estetään viitatun rivin poistaminen/päivittäminen ja tämä on oletusarvoinen määritys. Cascadella muutokset/poistot, jotka kohdistuvat viitattuun riviin, vyörytetään myös viittaaviin riveihin. Set defaultilla viittaava arvo korvataan oletusarvolla. Ja viimeisenä set null asettaa viitatun rivin perusavaimen päivityksen tai poistamisen myötä viittaavassa rivissä arvon muuttamisen NULL-arvoon. (Nieminen 2009, 80.)

Avainehdokas on uniikki attribuutti tai attribuuttien kokoelma, josta ei voida vähentää yhtäkään attribuuttia rikkomatta sen ainutlaatuisuutta. Jokaisella taululla tulee olla vähintäänkin yksi avainehdokas, sillä perusavain valitaan aina avainehdokkaiden joukosta. Avainehdokkaan lisääminen tapahtuu komennolla UNIQUE (attribuuttilista). (Hernandez 2000, 213; Nieminen 2009, 82.)

Relaatioihin voidaan myös lisätä eheyssääntöjä ja avainehdokkaita. Eheyssäännöt tarkistavat, että talletettava tieto on oikeanlaista. Tällainen tarkistus tapahtuu CHECK (ehto) -komennolla. Yksinkertaisimmillaan tällainen ehto voi olla vaikkapa CHECK (LaiteLkm > 0). (Nieminen 2009, 83.)

3.4.2 Indeksointi

Kun tietokanta on suunniteltu loppuun, saatetaan joskus todeta sen toimintanopeudessa joitakin puutteita. Vaikka tietokannan nopeuttamiseen on monia eri ratkaisuja, tehokkain kaikista on varmaankin indeksointi.

Indeksin vastine reaalimaailmassa on kirjan hakemisto, johon on listattu aiheet ja miltä sivulta ne löytyvät. Sen tarkoitus on toteuttaa perusavainta ja nopeuttaa kyselyitä. On jopa väitetty, että suurin syy tietokantojen hitauteen on indeksien puute. Miinuksena indekseissä on, että ne hidastavat lisäystä, poistoa ja päivittämistä, mutta tämä ei kuitenkaan ole enää nykyään useinkaan suuri ongelma. (Nieminen 2009, 94.)

Ari Hovi tarjoaa puutteellisen indeksoinnin havaitsemiseen hyvin tehokkaan keinon, jota kutsutaan nimellä ”karkea siivilä”. Siinä ohjelmoija esittää jokaisen SQL-kyselyn kohdalla itselleen kysymyksen: ovatko kaikki WHERE-lausekkeen sarakkeet varmasti samassa indeksissä? Jos vastaus tähän kysymykseen on ei, silloin lisätään olemassa olevaan indeksiin puuttuvat sarakkeet yksi kerrallaan. Tätä jatketaan kunnes saadaan tehostettua kyselyä riittävästi. Jos tämä ei riitä, niin lisätään sinne myös loput sarakkeet, aina SELECT-lausekkeessa esitettyihin sarakkeisiin asti. Tämä ratkaisu ei tietenkään ole täydellinen, mutta se tuottaa helposti tyydyttävän tuloksen eikä sen käyttö vaadi paljoa vaivaa ja onkin siksi jotain mitä kaikkien olisi hyvä tietää. Riskinä tietenkin on huonon indeksin luominen, mutta silloin voidaan tällä tekniikalla luotu indeksi vain poistaa. (Hovi ym. 2005, 214.)

4 TOTEUTUS

Siinä missä edellinen luku käsitteli tietokannan perustamista teorian puolelta, tämä luku käsittelee sitä käytännössä. Luvun alussa käsitellään ensiksi projektin sijaintia yrityksen tarvemaisemassa ja tämän jälkeen käydään nopeasti läpi eri järjestelmien välisen viestinnän toteuttaminen JSONilla. Tämän jälkeen käydään läpi tietokannan uudelleensuunnittelun vaiheet ja tämän kanssa koetut haasteet ja onnistumiset.

4.1 Tarvemäärittely

Opinnäytetyönä toiminut projekti oli osa laajempaa projektia, jossa oli tarkoitus toteuttaa koiranäyttelypalvelu. Tämä projekti rajautui tietokannan uudelleensuunnitteluun ja kahden järjestelmän välisen viestinnän toteuttamiseen. Koiranäyttelyt-tietokanta itsessään toimii enemmän tietovaraston tavoin. Vaikkakin rivimäärä on suhteellisen rajoittunut verrattuna joihinkin suuriin tietovarastoihin, niin tuhansien samanaikaisten raskaiden kyselyiden tuottama hidastuminen ja sen korjaaminen tuottavat haasteita kokemattomalle tietokantasuunnittelijalle niin kannan kuin kyselyidenkin kautta.

Järjestelmää on ollut kehittämässä ajan myötä useita henkilöitä, joten ratkaisut koodin ja tietokannan puolella olivat hyvin vaihtelevia ja eri ideologioihin perustuvia. Tietokannan alkutila ei ollut kehuttavalla tasolla: se sisälsi ylimääräisiä tauluja esimerkiksi ideoista, joita ei ikinä toteutettukaan; asia mihin allekirjoittanut myöntyy itsekin. Tämän lisäksi jotkin tauluista olivat paisuneet suuremmiksi kuin oli järkevää käsitellä. Viimeisenä herätyskellona toimi kannasta tapahtuvien hakujen hitaus, koska sovellus oli suuremman suoritustaakan alla kuin oltiin osattu olettaa ja täten aiheutti kiireellisiä ongelmia.

Projekti aloitettiin alustavana työnä kahden eri järjestelmän sulauttamiseksi yhteen, mutta tilanne on sittemmin muuttunut epävakaaksi. Tämän takia olen painottanut teoriaa huomattavan paljon verrattuna käytäntöön, jotta työ ja dokumentaatio mitä työstä syntyy voisi toimia apuna kokonaisprojektin ja tulevien vastaavien projektien kehityksessä.

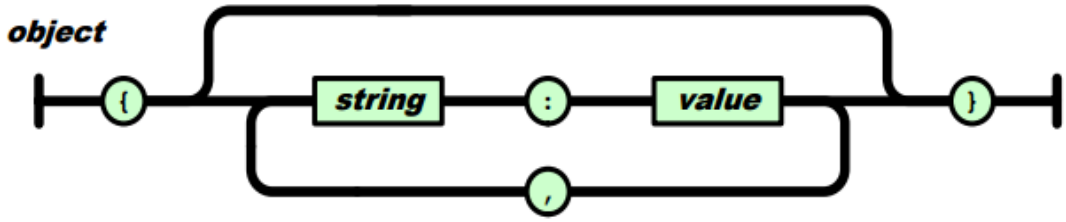
4.2 Järjestelmien välisen viestinnän toteuttaminen JSONilla

Koiranäyttelypalvelut koostuvat yhteensä kolmesta eri tuotteesta, joten näiden välillä on luonnollisesti tarve tiedonsiirrolle järjestelmästä toiseen. Tämä tiedonsiirto oli aiemmin tapahtunut Excel-tiedostojen avulla, jotka asiakas otti ensimmäisestä järjestelmästä ja lähetti yritykselle toiseen järjestelmään vietäväksi. Tästä tavasta aiheutui kuitenkin ajan myötä mittavia määriä ylimääräistä työtä. Manuaalinen tietojen tuonti tuotti lisätyötä sekä enemmän mahdollisuuksia virheelliseen tiedon muotoon, josta muistona eräskin pitkä ilta, jolloin allekirjoittanut pääsi korjaamaan itse aiheuttamaansa virhettä aamuyöhön asti.

Tämä järjestelmä päätettiin korvata väliaikaisella rutiinilla, jota käytettäisiin siihen asti, että eri järjestelmät saataisiin sulautettua yhdeksi. Rutiinin suoritustavaksi valittiin tiedonsiirto JSONilla, joka ajetaan tietokantaan vastaanottavassa järjestelmässä.

Järjestelmän toiminnan ideana on, että ilmoittautumispuolella ne tiedot, jotka ennen vietiin Excel-tiedostoon, viedään nyt JSON-tiedostoon olioiksi, jotka koostuvat avain-arvopareista. Seuraavalla sivulla oleva kuva havainnollistaa olioiden muotoa ja

luontia. Tämä tiedosto on järjestelmän käytettävissä ilmoittautumispalvelussa. Tämän jälkeen uuden näyttelyn perustaminen tapahtuu tulosjärjestelmässä napin painalluksella, jolloin tietokantaan viedään kaikki näyttelyn tiedot, kuten ajankohta ja koirien tiedot.



Kuva 15. JSON-olion tai -oliojoukon luonti (ECMA International 2013, 2.)

JSON-datan käsittelyyn ei valitettavasti ole Classic ASPissa tai VBScriptissä valmista rutiinia johtuen tekniikoiden elinikäeroista, mutta siihen on tehty muutamia avoimen lähdekoodin toteutuksia ajan myötä. Toteutus, jota käytin projektissa, on Gerrit van Kuipersin ASPJSON. Tämä toteutus pitää sisällään rutiineja JSON-tiedoston ASP-objektiin lukemiseen, päivittämiseen sekä tietojen lisäämiseen että poistamiseen ja on tekijänsä mukaan vapaasti käytettävissä. ASPJSONin alkuperäinen sivusto on kadonnut verkosta tämän projektin aikana, mutta siihen pääsee Wayback Machinen avulla. Suurimmat syyt valmiin toteutuksen käyttöönottoon ovat virheiden vähentäminen valmiiksi testatulla ohjelmistolla sekä ajan säästäminen siihen asti, kun uusi yhdistetty järjestelmä on valmis. (ASPJSON 2016.)

Näyttelyn ja koirien tiedot saadaan eriteltyä JSON-tiedostosta seuraavalla sivulla esitetyllä tavalla, jonka jälkeen ne ajetaan lopuksi tietokantaan. Tämän jälkeen uusi näyttely on valmis käyttöä varten. Prosessi itsessään ei ole monimutkainen, mutta tietojen manuaalisesti vieminen tuotti inhimillisiä virheitä ja tiedostomuotojen yhteensopimattomuus tuotti virheitä dataan, joka jouduttiin sitten tarkistamaan ihmisvoimin.

```
' Käydään koirien lista läpi ja lisätään jokaisen koiran tiedot kantaan.
For Each dogs In oJSON.data("ListOfDogs")
Set dogs = oJSON.data("ListOfDogs").item(dogs)

   sSql = sSql & "INSERT INTO DOGS (RegisterNumber, Breed, Class, Sex, Name) VALUES ("

   sSql = sSql & SetQuotesOrNull(dogs.item("RegisterNumber")) & ", "
   sSql = sSql & SetQuotesOrNull(dogs.item("Breed")) & ", "
   sSql = sSql & SetQuotesOrNull(dogs.item("Class")) & ", "
   sSql = sSql & SetQuotesOrNull(dogs.item("Sex")) & ", "
   sSql = sSql & SetQuotesOrNull(dogs.item("Name")) & ");"
Next
```

Kuva 16. Yksinkertaistettu ja lyhennetty versio koiratietojen tuontirutiinista

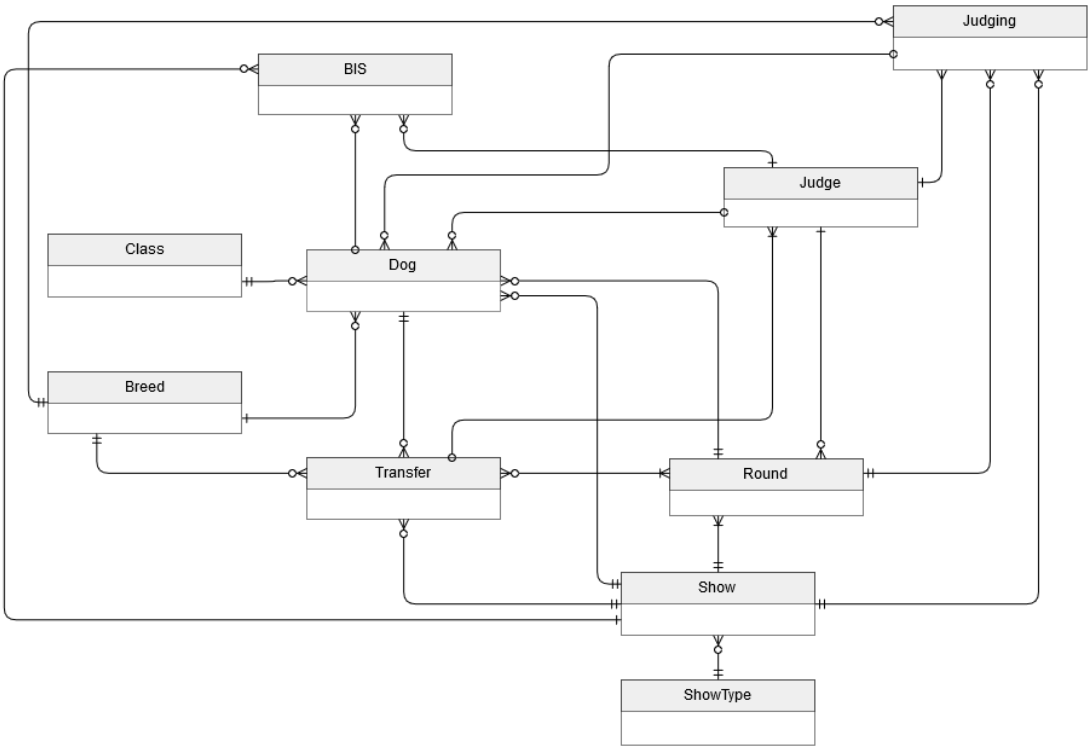
4.3 Käsiteanalyysi

Kuten teoriapuolella mainitsin, ensimmäinen vaihe tietokannan suunnittelussa on luoda siitä korkean abstraktion tason mukainen malli, jonka päälle myöhemmät vaiheet rakentuvat. Normaalisti tämä tapahtuisi asiakkaan kanssa tapahtuvan keskustelun kautta, mutta koska tällä kertaa kyseessä oli jo olemassa ja käytössä olevan järjestelmän uudelleensuunnittelu, lähdin lyhyen työtovereiden kanssa käymäni keskustelun jälkeen luomaan koiranäyttelytietokannasta ER-kaaviota, käyttäen harakanvarvasnotaatiota sen merkitsemiseen.

Halusin käydä koko prosessin läpi, jotta voisin sanoa, että olin tehnyt sen ”kuten kirjassa käsketään”, mutta sain huomata, ettei käsitemallin tekeminen edes olemassa olevasta tietokannasta onnistu ihan käden käänteessä. Tämän mallin tekemiseen käyttämäni aika ei ollut hukkaan heitettyä. Pystyin sen tekemisen yhteydessä selvittämään, mitä ylimääräisiä tauluja kantaan oli kerääntynyt ja mitä tauluja kannattaisi ehkä yhdistää. Tein myös joitakin lisäyksiä perustaen ne keskusteluun, jonka kävin aiemmin koiranäyttelyprojektia eteenpäin vieneen työtoverin kanssa, jotta tulevaisuuden tarpeet olisivat hiukan paremmin täytetyt. Tärkeintä kuitenkin oli saada aikaiseksi tietokanta, joka kestäisi ja johon olisi helppo tehdä muutoksia ja päivityksiä muuttuvien tarpeiden ja toiveiden mukaan.

Sain aluksi rajattua työssäni hyödynnettävien taulujen määrän 15 tauluun, joista lähdin luomaan yksilötyyppejä käsitemalliin. Loput kannan taulut olivat sivuston kannalta hallinnollisesti tärkeitä, mutta eivät työtäni koskettavia. Seuraavalla sivulla olevasta kuvasta on nähtävissä, millaisen käsitemallin sain luotua. Vaikka siinä esite-

täänkin vain kymmenen yksilötyyppiä ilman ominaisuuksia, on tämä kymmenen jo itsessään karsittu ja mielestäni paranneltu versio alkuperäisestä. Se myös auttoi yksinkertaistamaan käsitemallia, jonka alkuperäinen muoto oli hyvin kaoottinen.



Kuva 17. Tietokannan uudelleensuunnittelun alussa syntynyt käsitemalli

Vaikka suurin osa tauluista kääntyikin sujuvasti yksilötyypeiksi, aiheutti esimerkiksi Transfer-taulun muuntaminen huomattavasti ongelmia. Tämä taulu koostuu muun muassa kahdesta viittauksesta Judge-tauluun ja kahdesta viittauksesta Round-tauluun enkä aivan ongelmitta osannut kuvata tätä suhdetta sillä tavalla, millaisena halusin taulun loppujen lopuksi pitää. Suurin osa näistä on onneksi hioutunut pois, mutta tämä prosessi itsessään toimi hyvin kokemuksena siitä, ettei suunnitteluprosessi aina mene pelkästään yhteen suuntaan, vaan sen sijaan toimii kuin saha, leikaten hitaasti kohti sitä täydellisintä lopputulosta.

4.4 Fyysinen suunnittelu

Edellisessä vaiheessa loin käsitemallin joka toimii yleisenä karttana, mutta joka on liian abstrakti käytännön työhön. Nyt kun yksilötyypeistä ja suhteista luodaan relaatioita, lisään näihin attribuutit näkyviin, koska ne ovat nyt huomattavasti hallitta-

vammassa olomuodossa. En listannut niitä käsitemalliin, koska jo valmiiksi epäselvässä käsitemallissa, kuten se alun perin oli, kymmenien attribuuttien lista on omiaan vain vaikeuttamaan suunnittelua ja tekemään siitä vaikeampilukuisen.

Kuten taulujen kohdalla aiemmin, kävin attribuutteja lisätessäni läpi, mitä niistä voidaan tiputtaa pois puhtaasti ylimääräisinä. Tämän lisäksi lisäsin muutamia, joita järjestelmän oletetaan tarvitsevan tulevaisuudessa. Yhdenmukaistin myös nimeämistapaa, jotta lopputuloksena syntyvästä tietokannasta olisi helpompi selvittää, mitä yksittäiset arvot oikeasti ovat. Tämän vaiheen tuotoksesta on alapuolella esimerkkikuva, missä on nähtävillä osa DOG-relaation attribuuteista täyden listan ollessa vielä tässä vaiheessa turhan pitkä esitettäväksi. Tämän lisäksi BREED-relaatio on kokonaisuudessaan nähtävillä.

```
DOG (                    BREED (
    Uid NOT NULL,            Uid NOT NULL,
    Order,                   Name_fi,
    Register_number,         Name_en,
    BreedId,                 Renki_code,
    ClassId,                 FCI,
    Sex,                     FCI_group,
    Name,                    PRIMARY KEY (Uid))
    Owner,
    Owner_address,
    Owner_city,
```

Kuva 18. Relaatioiden muodostuminen, viiteavaimia tai arvojoukkoja ei ole vielä määritelty tässä vaiheessa

Varmistettuani, että kaikilla yksilötyypeillä oli perusavaimet, ryhdyin purkamaan yksilötyyppejä relaatioiksi. Näiden jälkeen piti purkaa monen-suhde-moneen -yhteydet, joita tässä käsitemallissa oli vain yksi, Transfer-yksilötyypin suhde Round-yksilötyyppiin. Purin tämän yhteyden ja käytin huomattavan ajan sen pohtimiseen, mitä sen kanssa pitäisi oikein tehdä. Loppujen lopuksi jätin sen hyvin pitkälti samaan muotoon, missä se oli vanhassakin kannassa ollut. Moniarvoisten attribuuttien pur-

kaminen relaatioiksi tuotti huomattavan määrän lisää relaatioita ja kevensi puolestaan muutamia raskaita yksilötyyppejä.

Tämän jälkeen muodostin viiteavaimet olemassa olevista yhteyksistä. Tämä onnistui rutiininomaisesti eikä tuottanut mitään ongelmakohtia. Sitten lisäsin jokaiselle attribuutille tälle sopivan arvojoukon SQL Serverin peruskokoelmasta. Tässä vaiheessa erottelin joitakin attribuutteja, jotka eivät tulisi jäämään normalisoinnin haaviin kiinni, omiksi relaatioikseen, jotta näiden käyttäminen helpottuisi.

4.5 Normalisointi

Tässä kohtaa osa relaatioista on normalisoimattomia eli UNF-muodossa. Työtä on tavallista vähemmän, koska tietokanta sisälsi valmiiksi hyviä rakenteita, jolloin uudelleensuunnitteluun ei ole tullut aivan valtavaa määrää normalisoitavia relaatioita. Kuten teoriapuolellakin, tietokanta normalisoidaan 3. normaalimuotoon asti ja sen jälkeen tarkistellaan tarvetta denormalisoida 2. normaalimuotoon kyselyiden nopeuttamiseksi. Koiranäyttelytietokanta on kuitenkin pääasiassa kyselyihin vastaava palvelu, jolloin nopealla talletuksella ei itsessään saavuteta niin suurta etua kuin nopeilla kyselyillä.

Relaatioille suoritetaan normalisointi ensimmäiseen normaalimuotoon, 1NF:iin, joka tuottaa useita uusia relaatioita, kuten alla kuvatun DOG_TITLE:n. Title oli DOG relaation attribuutti, jolle oli tallennettu monta arvoa per monikko, jolloin se täytyi muuttaa omaksi relaatiokseen.

```
DOG_TITLE (
DogId DOMAIN (INT) NOT NULL,
Title DOMAIN (VARCHAR(25)) NOT NULL,
PRIMARY KEY (DogId, Title),
FOREIGN KEY (DogId) REFERENCES DOG)
```

Kuva 19. Esimerkki 1NF:n toteuttamisesta

Toisen normaalimuodon toteuttaminen ei tuottanut juurikaan ongelmia näiden relaatioiden kanssa. Relaatiot itsessään olivat hyvin suositun ratkaisun mukaisia, joka tekee 2NF:n mukaisen kannan tekemisestä hyvin helppoa eli perusavaimena oli hyvin usein surrogaatti eli tietokantahallintajärjestelmän ylläpitämä juokseva luku. Ne relaatiot, joissa kuitenkin perusavain koostui useasta attribuutista, olivat kaikki valmiiksi 2NF-muodossa, koska ne olivat johdettuja aiemmasta datasta normalisoinnin yhteydessä.

Relaatioiden kolmannen normaalimuodon toteuttamisen yhteydessä nousee muutamia tapauksia esille. On kuitenkin järkevä ajatella, että ne denormalisoidaan, sillä tapaukset vastaavat pääosin samaa tilannetta kuin postinumeron ja postitoimipaikan suhde. Tämän lisäksi ainakin yhdessä tapauksessa taulun jakaminen aiheuttaisi tulevaisuudessa yhden liitoskyselyn lisää suurimpaan osaan kyselyistä, jolloin on tehokkaampaa pitää tunnistus surrogaattiin nojaavana. Lisäksi tämä säästää huomattavasti päänvaivaa suunnitelman käyttöönottajalta, koska tämän ei tarvitse keksiä kirjausaikaa kymmenille tuhansille koirille, joille arvoa ei olla vielä ikinä merkitty.

Tämän vaiheen lopussa on saavutettu relaatioiden suhteen joidenkin hyvin raskaiden relaatioiden keventämistä ja harvemmin tarvittuja attribuutteja on saatu siirrettyä omiin relaatioihin.

4.6 Viimeistely

Viimeistelyvaiheessa relaatioihin lisätään vielä NOT NULL -määritykset niihin attribuutteihin, jotka ovat pakollisia ja joista tämä vielä puuttuu. Tämän lisäksi tulee päättää viiteavainten poisto- ja päivityssäännöistä, mutta niitä ei vaihdeta, koska mielestäni oletusarvoinen ON UPDATE/DELETE RESTRICT on juuri se, mitä näiden osalta kaivataan.

Nyt jäljellä on vain relaatioiden muuttaminen SQL-luontilausekkeiksi ja dokumentaatio. Dokumentaationa tehdään tauluista taulukuvaukset, jotka toimitetaan työn tilaajalle relaatioiden listan ja SQL-luontilausekkeiden kanssa. Käytin dokumentaatioiden tekemiseen pohjana Niemisen tekemää mallia sen selkeyden vuoksi. Liitteet 1-

3 kuvaavat millaisia nämä tiedostot tulevat olemaan. Näissä on käytetty esimerkkinä Round-relaatiota. (Nieminen 2009, 99.)

Tämän jälkeen työn tilaajalla on kaikki tarpeelliset kuvaukset ja alustavat SQL-lauseet uuden kannan luomiseksi. Se mitä tämän jälkeen pitäisi tehdä, on siirtää kaikki tiedot vanhasta kannasta uuteen ja ottaa tämä uusi kanta käyttöön vanhan tilalle, mutta se jää suunnitelman toteuttajan, työn tilaajan, vastuulle.

5 LOPPUSANAT JA YHTEENVETO

Tietokanta on datapainotteisen tietojenkäsittelyprojektin sydän. Tämä ajatus jäi mieleeni hyvin voimakkaana projektin päättyessä. Vaikka muut osat tietojenkäsittelyprojektia ovatkin tärkeitä, niin ei datapainotteinen projekti jaksa juosta kovaa, kun sen sydän lyö huonosti. Silloin on hyvä hoitaa sitä indeksoinnilla ja uudelleensuunnittelulla.

Projekti tuli osakseni osittain siksi, etten ollut tajunnut ajoissa miettiä sopivaa aihetta opinnäytetyöhön. Kun kysyin ehdotuksia eräältä työtoveriltani, ehdotti hän koiranäyttelytietokannan uudelleensuunnittelua. Tästä tarpeesta olin jo lähes puoli vuotta valittanut, että se pitäisi tehdä. Aihe osoittautui, ilman suurempaa yllätystä, hyvin vaikeaksi aiheeksi, sillä tietokannat eivät ole olleet missään vaiheessa helppo aihe minulle. Lähdin suorittamaan tätä projektia lukemalla mahdollisimman paljon kirjallisuutta tietokannoista. Mitä enemmän luin, sen enemmän aloin ymmärtää aiheen olevan paljon laajempi, mitä olin osannut olettaakaan.

Projektin kaikkein aikaa vievimmät osat tuntuivat olevan sellaisia, jotka veivät vähiten tilaa paperilla eli suunnittelutyö. Luin ja kirjoitin huomattavasti enemmän tekstiä relaatioista ja niiden suhteista tai normalisoinnista ja sen toteuttamisesta kuin tietokannan suunnittelusta, mutta silti työstä meni eniten aikaa ja aivovoimaa sellaiseen osaan, joka ei edes erotu paperilla erityisen työlääksi.

Myöskään kiireet työssä ja sen jälkeen työsuhteen loppuminen ja projektin tulevaisuuden epävarmuus eivät olleet omiaan voimistamaan motivaatiota. On vaikeaa suunnitella kantaa, jos ei ole varma mihin suunnitelmaa oikeastaan tullaan ikinä käyttämään. Tästä syystä tämä opinnäytetyö painottaa teoriaa niin paljon kuin se tekee, koska vaikka tätä suunnitelmaa ei koskaan käytettäisi, niin olisi kuitenkin arvokasta työpaikalle, että tietokantojen suunnittelusta olisi lyhyt ja nopeasti luettavissa oleva yhteenveto. Itse olisin sitä kaivannut!

Eli kun on luettu ja kirjoitettu teoriaa, tehty käytännön suunnitelma ja viety se toteutusta vaille valmiiksi, niin mitä saavutettiin? Tulokseksi saatiin dokumentaatiot, joiden avulla voidaan projekti toteuttaa alusta loppuun asti ja selventää ratkaisujen takana olevia ideoita.

LÄHTEET

Technopedia. Active Server Pages (ASP pages). Viitattu 4.10.2016. Saatavissa: https://www.techopedia.com/definition/23088/active-server-pages-asp-pages

Microsoft. 2016. Viitattu 4.10.2016. Saatavissa: https://support.microsoft.com/en-us/kb/2669020

Tutorialspoint 2016. Viitattu 4.10.2016. Saatavissa: http://www.tutorialspoint.com/vbscript/vbscript_overview.htm

ECMA International 2013. The JSON Data Interchange Format. Viitattu 15.10.2016. Saatavissa: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

ASPJSON, Wayback Machinen kautta 2016. Viitattu 15.10.2016. Saatavissa: https://web.archive.org/web/20160308120309/http://aspjson.com/

Smiley, J., 2015. Viitattu 15.10. 2016. Saatavissa: http://www.johnsmiley.com/visualbasic/vbhistory.htm

Popovic, J. 2015. JSON Support in SQL Server. Viitattu 15.10.2016. Saatavissa: https://blogs.msdn.microsoft.com/jocapc/2015/05/16/json-support-in-sql-server-2016/

Microsoft. 2014. End of Mainstream support for SQL Server 2008 and SQL Server 2008 R2 Viitattu 15.10.2016. Saatavissa: https://blogs.msdn.microsoft.com/sqlreleaseservices/end-of-mainstream-support-for-sql-server-2008-and-sql-server-2008-r2/

Date, C. J., Kannan, A. & Swamynathan, S. 2003. An introduction to Database Systems. 8. versio. Pearson.

Hernandez, Michael J. 2000. Tietokannat – suunnittelu ja toteutus. Jyväskylä: Gummerus Kirjapaino Oy.

Nieminen, Hans. 2009. Tietokannan suunnittelu.

Hovi, A., Huotari, J. & Lahdenmäki, T. 2005. Tietokantojen suunnittelu ja indeksointi. Porvoo: WS Bookwell.

Computer Hope:n sivut. Dynamic website. Viitattu 14.11.2016. Saatavissa: http://www.computerhope.com/jargon/d/dynasite.htm

Mar, W. 2016. Viitattu 14.11.2016. Saatavissa: http://www.wilsonmar.com/1asp.htm

Lee, J. 2013. Viitattu 14.11.2016 Saatavissa: https://blog.udemy.com/oracle-vs-mysql-vs-sql-server/

TAULU: ROUND

Perusavain: Uid

Viiteavaimet

Viiteavain	Viitattava taulu	Poistosääntö	Päivityssääntö
Judge	JUDGE	esto	esto
ShowId	SHOW	esto	esto

Perusavaimeen viittaavat taulut

Viittaava taulu	Viittaavan taulun viiteavain
TRANSFER	Round_old
TRANSFER	Round_new
JUDGING	RoundId

Sarakkeet

Nimi	Arvojoukko (Tietotyyppi)	Pakollinen	Oletusarvo	Selitys
Uid	INT	tosi	Surrogaatti	Taulun perusavain
Round_number	INT	tosi		Kehän numero
Comp_date	DATE	tosi		Kehän päiväys, tunniste monipäiväiselle näyttelylle
Judge	INT	tosi		Viittaus kehän tuomariin
Show	INT	tosi		Näyttelyn tiedot

Eheyssäännöt

Nimi	Sääntö
PK_ROUND	PRIMARY KEY (Uid)
FK_ROUND_JUDGE	FOREIGN KEY (Judge) REFERENCES JUDGE
FK_ROUND_SHOW	FOREIGN KEY (Show) REFERENCES SHOW

```
DOMAIN ID           INT
DOMAIN KEHA         INT CHECK (VALUE BETWEEN 1 AND 9)
DOMAIN KIERROS      INT CHECK (VALUE > 0)
DOMAIN KISA_PVM     DATE


ROUND (
    Uid DOMAIN (ID) NOT NULL,
    Round_number DOMAIN (KEHA) NOT NULL,
    Comp_date DOMAIN (KISA_PVM) NOT NULL,
    Judge DOMAIN (ID) NOT NULL,
    Show DOMAIN (ID) NOT NULL,
    PRIMARY KEY (Uid),
    FOREIGN KEY (Judge) REFERENCES JUDGE,
    FOREIGN KEY (Show) REFERENCES SHOW)
```

```
CREATE TABLE ROUND (
    Uid INT NOT NULL,
    Round_number INT NOT NULL,
    Comp_date DATE NOT NULL,
    Judge INT NOT NULL,
    Show INT NOT NULL,
    CONSTRAINT PK_ROUND PRIMARY KEY (Uid))

ALTER TABLE ROUND
    ADD
    CONSTRAINT FK_ROUND_JUDGE
      FOREIGN KEY (Judge) REFERENCES JUDGE,
    CONSTRAINT FK_ROUND_SHOW
      FOREIGN KEY (Show) REFERENCES SHOW,
    CONSTRAINT DM_ROUND_KEHA
      CHECK (KEHA BETWEEN 1 AND 9),
    CONSTRAINT DM_ROUND_KIERROS
      CHECK (KIERROS > 0)
```