OULUN SEUDUN
AMMATTIKORKEAKOULU

Miika Kontio

TIETOKANNAN KEHITTÄMINEN KODINMONITOROINTIJÄRJESTELMÄN SENSORIMITTAUKSILLE

TIETOKANNAN KEHITTÄMINEN KODINMONITOROINTIJÄRJESTELMÄN SENSORIMITTAUKSILLE

Miika Kontio
Opinnäytetyö
Kevät 2013
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehitys

Tekijä: Miika Kontio
Opinnäytetyön nimi: Tietokannan kehittäminen kodinmonitorointijärjestelmän sensorimittauksille
Työn ohjaaja: Eero Nousiainen
Työn tilaaja: Oy PremiSense Ltd
Tilaajan yhteyshenkilö: Jussi Vepsäläinen
Työn valmistumislukukausi ja -vuosi: Kevät 2013
Sivumäärä: 77 + 8 liitettä

Tämän työn aiheena määrittää, suunnitella ja toteuttaa tietokanta kodinmonitorointijärjestelmän sensorimittauksille Oy PremiSense Ltd -yrityksen tarpeisiin. Työn tavoitteena oli saada aikaan pääohjelman käyttöliittymän kanssa toimiva ohjelmistoon sisällytetty tietokanta. Tietokannaksi valittiin SQLite. Vertailun vuoksi työssä kokeiltiin MySQL- ja Microsoft SQL Server CE-tietokantajärjestelmiä.

Pääohjelman ohjelmointiin käytettiin C#-ohjelmointikieltä ja sitä toteutettiin Microsoft Visual Studion 2012 Express -versiolla. SQLite-tietokantaa hallinnoitiin SQLite Database Manager -ohjelmalla. SQLite-tietokannan suunnitteluun käytettiin MySQL Workbench -ohjelmaa. Koodin integrointiin käytetiin WinMerge-ohjelmaa.

Työssä käytettiin ketterää Scrum-tuotekehitysmallia ohjelmistosovelluksen projektin aikataulutukseen, ominaisuuksien suunnitteluun ja valmistumisen seurantaan.

Työ onnistui hyvin ja tietokanta saatiin luotua ja otettua käyttöön. Työhön kuului myös tiedon jatkokäsittely eli esimerkiksi graafisten kuvaajien piirto tietokannasta haetulla datalla. Tämäkin tavoite saatiin työssä toteutettua. Kolmantena osa-alueena toteutukseen kuului itse pääsovelluksen käyttöliittymän toteutus. Tietokannan optimointi ja osittaminen jäivät vähälle, ja niinpä ne voidaan lukea mahdollisiin kehitys- ja jatkotoimenpiteisiin.

Asiasanat: tietokannat, SQLite, sensorit, C#

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Software Development

Author: Miika Kontio
Title of thesis: Database for sensor data
Supervisor: Eero Nousiainen
Customer: Oy PremiSense Ltd.
Customer's contact: Jussi Vepsäläinen
Term and year when the thesis was submitted: Spring 2013
Pages: 77 + 8 appendices

The idea of this thesis was to determine, design and implement a database for sensor measurements of a home monitoring system for the company called Oy Premisense Ltd. The goal was to get a fully functional database embedded in to the main software. For the database was chosen SQLite. For comparison also MySQL- and Microsoft SQL Server CE -database systems were tried.

Main program was programmed in C#-programming language and it was developed in Microsoft Visual Studio 2012 Express edition. SQLite-database was controlled in SQLite Database Manager -software. Database design was done in MySQL Workbench. Integration of codes was done in WinMerge.

In this thesis was used an agile project managering system called Scrum. Scrum was used in time table design, planning the features of the software and looking up results.

The thesis work went well and database was created and embedded in the main software. The job also included further data handling, for example drawing of the graphical plots with data that was got from database. This goal was also successfully done in the work. Third area of implementation was to take part of the creation of the graphical user interface for the main software. Optimization and partitioning of the database were less successfully done and therefore they can be included in the possible future versions of the software.

Keywords: database, sqlite, sensor, C#

SISÄLLYS

SANASTO

API = Application Programming Interface eli sovelluksen ohjelmointirajapinta

Data = tietoa sisältävä kokonaisuus. Esimerkiksi sensoridata voi sisältää sensorin lähettämän lämpötilan tai ohjelmiston mittaaman kellonajan.

DataSet = C#-ohjelmointikielen komponentti, joka toimii SQLite-tietokannan ja C#-ohjelmakoodin välisenä rajapintana ja datan sijoituspaikkana.

DataTable = DataSet voidaan pilkkoa DataTableiksi, jotka vastaavat yhtä tietokantataulua

Debuggaus = koodin suorituksen seuraaminen ja sen avulla virheen etsiminen.

DLL = Dynamic Link Library eli dynaaminen linkattava kirjasto johon sisältyy paljon valmista ohjelmakoodia, ja joka voidaan ottaa Visual Studiossa käyttöön lisäämällä DLL referenssiksi.

Flat-file-tietokanta = yleensä tekstitiedostossa oleva tietorakenne, jossa tieto sijaitsee riveittäin.

Funktio = metodi, koodissa jokin toistuva tehtävä voidaan kirjoittaa funktioksi, jota voidaan kutsua ilman että koodia tarvitsee kirjoittaa enää kokonaan uudelleen.

Hardware = fyysinen laitteisto. Se voi olla esimerkiksi sensori, tietokone tai ledi.

Integrointi = eri ohjelmakoodien liittäminen yhteen yhdeksi kääntyväksi ja toimivaksi kokonaisuudeksi.

Kenttä = tietokannan yksi sarake.

Luokkakirjasto = ohjelmointikieleen liittyvä suuri joukko valmiita ohjelmaluokkia, joita voi käyttää joko luomalla niistä olion tai suoraan staattisesti.

Ohjelma = Tietokoneohjelmakoodi, joka on koottu tiettyjä ominaisuuksia sisältäväksi toimivaksi kokonaisuudeksi. Ohjelma voi olla tekstimuotoinen konsolisovellus tai graafinen sovellus.

Olio = Olio-ohjelmoinnissa käytettävä kokonaisuus. Oliot sisältävät toisiinsa loogisesti liittyvää tietoa ja toiminnallisuutta. Olioita voidaan luoda C#-ohjelmakoodissa new-operaattorilla.

Parametri = tieto joka välitetään funktiolle esimerkiksi string- tai DataSet-tyyppisessä muodossa. Funktiossa käytetään parametria erilaisten toimintojen toteuttamiseen.

Pseudokoodi = Koodi, joka on tarkoitettu lähinnä hahmottamaan sanallisesti koodin toimintaa ilman, että se toteuttaa valitun ohjelmointikielen mukaista syntaksia.

Sensori = eli anturi on fyysinen laite, joka mittaa ympäristöstä jotain suuretta.

Tietue = tietokannan yksi rivi.

UI = User Interface eli käyttöliittymä on graafinen esitys siitä, mitä käyttäjä näkee ohjelmistosta. Käyttöliittymä sisältää informaatiota tekstinä, kuvina ym. ja käyttäjä voi antaa esimerkiksi painonapeilla käskyjä, mitä ohjelman pitää tehdä (esimerkiksi vaihtaa sivua tai poistua ohjelmasta).

Visual Studio = Microsoftin ohjelmointiympäristö, joka oli tämän työn tärkein työkalu.

1 JOHDANTO

Tämän työn tavoitteena oli määritellä, suunnitella ja toteuttaa tietokanta Oy PremiSense Ltd:lle sensorien mittaamaa dataa varten. Tietokannan tulisi täyttää annetut kriteerit. Lisäksi tietokannan tietoa tulisi jatkokäsitellä sopivalla tavalla ja esittää graafisessa muodossa esimerkiksi pylväs- tai viivadiagrammeina. Tiedonhaku, tiedon jatkokäsittely sekä graafinen esitys koodattiin C#-ohjelmointikielellä. Tämän työn koodit on kommentoitu siten, että niiden ymmärrettävyys olisi mahdollisimman hyvä.

Tietokannaksi valittiin SQLite. Perustelu valintaan löytyy luvusta 6. Tämän työn tekemisessä käytettiin vain ja ainoastaan ilmaisia vapaaseen lähdekoodiin perustuvia tietokoneohjelmia, koska ne toimivat oikeinkäytettyinä yhtä hyvin kuin kaupalliset vastineet.

PremiSense on vuonna 2012 aloittanut alle 10 henkilön startup-yritys, jonka tuote toteutetaan pilottihankkeena Oulun Bussiness Kitchenin tiloissa. Tuote koostuu fyysisistä laitteista ja ei-fyysisestä tietokoneohjelmasta. Laitteisto sisältää erilaisia sensoreita ja keräilylaitteen, jota kutsutaan dataloggeriksi.

Sensorit eli anturit muodostavat keskenään langattoman verkon, mittaavat ympäristöstä arvoja ja lähettävät mittaustuloksia langattomasti keräilylaitteelle tietyin väliajoin(1). Tieto siirretään keräilylaitteelta pc-koneelle synkronoitavaksi esimerkiksi USB:n avulla ja esitetään tietokoneohjelmalla, joka nimettiin HomeMonitoriksi. Tietokanta on pc-koneen ja HomeMonitorin välisenä rajapintana tiedon käsittelyssä.

PremiSensen kehitteillä olevan tuotteen ideana on mahdollistaa pienkiinteistönhallinnan erilaisten parametrien seuranta. Tässä työssä yhtenä esimerkkinä parametreista käytetään lämpötilamittauksia.

2 KÄYTETYT TYÖKALUT JA TEKNIIKAT

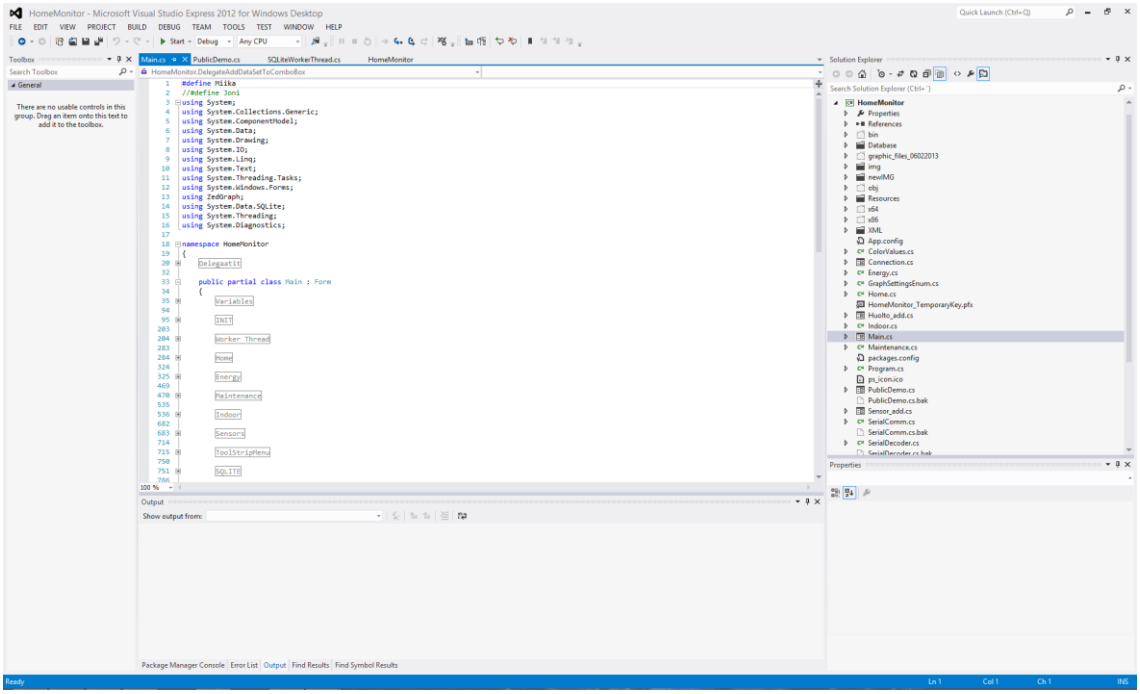
2.1 Microsoft Visual Studio 2012 ohjelmiston tuottamiseen

PremiSensen tuottaman pienen käyttäjäkyselyn mukaan noin 80 % vastanneista 36 asiakkaasta käytti aktiivisesti Windows-pöytäkonetta. Tämä tutkimus tuki aiempaa yrityksen käsitystä Windows-koneiden yleisyydestä. Ohjelmiston tuottamiseen eli koodin kirjoittamiseen, kääntämiseen, debuggaamiseen ja julkaisuversion luomiseen valittiin ohjelmointiympäristön nimeltä **Microsoft Visual Studio 2012 Express C# For Desktop**.

Visual Studio on Microsoftin ohjelmointiympäristön yleisin työkalu. Tässä työssä käytetty C# Express -versio on ilmainen, vaikkakin täysistä ominaisuuksista riisuttu, ja sen koodikielenä on C#. Työkalu on erityisesti tarkoitettu stand alone -työpöytäsovelluksien luomiseen Windows käyttöjärjestelmille. (2.)

2.1.1 Visual Studion käyttöliittymä

Visual Studion käyttöliittymä koostuu useista ikkunoista. Keskellä ruutua iso valkoinen alue on pääikkuna, jossa näkyy valittu lähdekooditiedosto. Vasemmalla on työkalulaatikko (engl. toolbox), josta voidaan lisätä suunnittelunäkymässä (engl. designer) kontrolleja, joita on esimerkiksi tekstikentät, painikkeet, paneelit ja valikot. Designerilla tehdään siis ohjelman User Interface (UI) eli käyttöliittymä. Oikealla on Solution Explorer -ikkuna, jossa näkyy projektin sisältämät tiedostot, ja sen alla on Properties-ikkuna, jossa näkyy valitun kontrollin tarkempia tietoja. (Kuva 1.)



KUVA 1. Visual Studio 2012 -käyttöliittymä, jossa on avoinna Main.cs-lähdekooditiedosto

2.1.2 .NET Framework

Visual Studiolla tuotetut ohjelmat käyttävät .NET Framework -alustaa. Alusta tukee 20 eri ohjelmointikieltä, joista yleisin on tässä projektissa käytetty C#. Siinä on oma ajoympäristö CLR (Common Language Runtime) ja isot valmiit luokkakirjastot. CLR:n vuoksi C# ei ole natiivi ohjelmointikieli, koska C#:a ei voida ajaa ilman CLR:ää. Luokkakirjasto tarkoittaa valmiita koodeja, joita saa vapaasti käyttää, eli kaikkea ei tarvitse toteuttaa aina itse uudestaan. (3.)

Työssä valittiin käytettäväksi .NET Framework 4.0:n, koska sillä voi luoda visuaalisia Windows lomakeohjelmia (engl. form application) sekä 32-bittiseen että 64-bittiseen käyttöjärjestelmään (4).

Lomakeohjelma tarkoittaa tavallista Windowsissa käytettävää ikkunamaista sovellusta, jossa on graafinen ulkoasu yläpalkin valikoineen, nappuloineen, kuvineen ja teksteineen. Lomakkeisuus on yksi .NET Frameworkin osa-alueista. (3.)

2.1.3 C#-ohjelmointikieli

C# on C-kielestä johdettu ohjelmointikieli, jossa yhdistyy C-kielen rakenne, C++-kielen tehokkuus ja Java-kielen helppous. C# on kehitetty Microsoftin toimesta vuonna 2000, ja jo vuonna 2003 se sai oman kansanvälisen ISO-standardinsa. (5.)

C-kieli ja sen johdannainen C# ovat käännettäviä ohjelmointikieliä. C#:lla kirjoitettu lähdekoodi siis käännetään Visual Studion omalla kääntäjällä (engl. compiler) tekstimuodosta tietokoneen ymmärtämään objektimuotoiseksi ohjelmaksi konekielelle (binääriksi eli ykkösiksi ja nolliksi). Tähän objektimuotoiseen ohjelmaan linkitetään valmiit kirjastomoduulit ja tuloksena syntyy suorituskelpoinen ohjelma. Ohjelma suoritetaan tavallisesti .exe-tiedostosta. (6.)

Ohjelmakoodin suoritus siirtyy käännetyssä C#-koodissa tavallisesti ylhäältä alas, mutta koodin suoritus ei siirry aina täysin suoraviivaisesti, koska se voidaan siirtää esimerkiksi funktion sisälle suorittamaan jokin toimitus (esimerkiksi lasku- tai sijoitusoperaatioita) tai esimerkiksi luuppiin eli toistorakenteeseen. Koodin suoritus jää pyörimään annetun kierrosmäärän ajaksi luupin sisälle.

2.1.4 Debuggaus

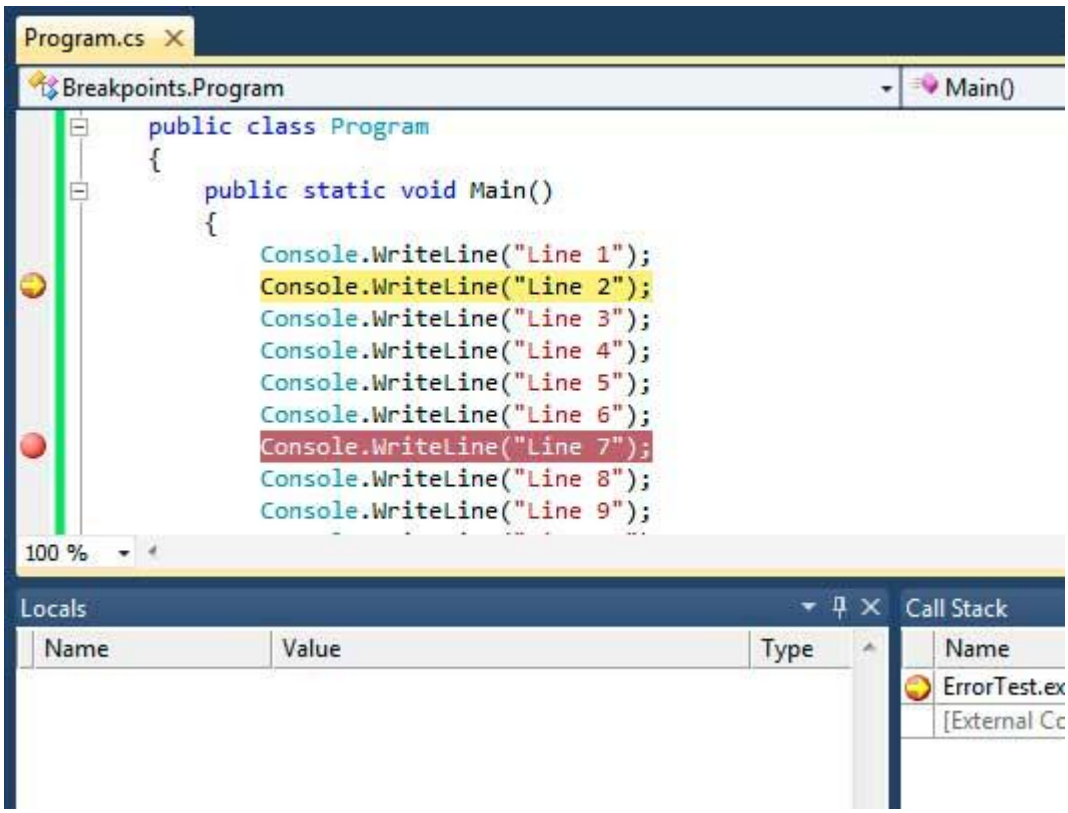
Kääntäjä ilmoittaa mahdollisista käännösvirheistä. Ohjelmoijan pitää korjata virheet, ennen kuin ohjelma on toimintakuntoinen ja ajovalmis. Jos koodi kääntyy, mutta toimii silti virheellisesti, ohjelmoija voi koettaa paikallistaa ja korjata virheet debuggauksen avulla. Debuggaaminen hoituu joko tulostamalla Visul Studion debuggerin output-ikkunaan tulostuksia tietyiltä riveiltä tai asettamalla halutuille riveille pysäytyskohtia (engl. break points). (7.)

Debuggerin output-ikkunaan tulostetaan C#-koodissa työpöytäohjelmassa rivi:

System.Diagnostics.Debug.WriteLine(”tämä on viesti”);

Pysäytyskohtien avulla voidaan helposti nähdä vaikkapa if-else-lauseessa, kumpaan haaraan koodin suoritus siirtyy tai mikä on muuttujien eli jonkin arvon

varastoivan yksikön sisältö ennen ja jälkeen pysäytyskohdan. Pysäytyskohdista päästään jatkamaan askel kerrallaan aina seuraavaan pysäytyskohtaan näppäimistön F5-näppäintä painamalla. Debuggaamalla on hyvin helppo nähdä, missä koodin suoritus etenee ja paikallistaa virhetilanteet. Kuvassa 2 break pointeja on laitettu koodiriveille 2 ja 7. Koodin suoritus on pysähtyneenä rivillä 2, kunnes debuggaaja painaa F5 jolloin koodin suoritus siirtyy taas ja pysähtyy riville 7.



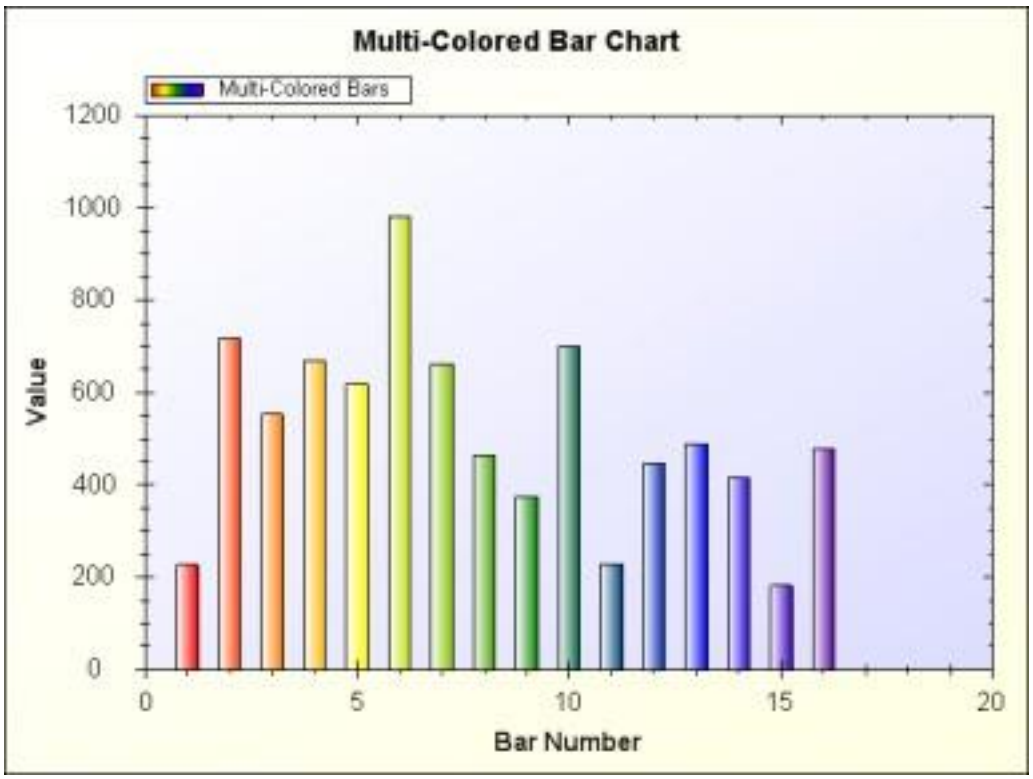
KUVA 2. Pysäytyskohtien käyttöä debuggaamiseen Visual Studiossa

2.1.5 ZedGraph

Graafit eli diagrammit eli kuvaajat tarkoittavat tiedon graafista esittämistä tietyin menetelmin. Graafissa on x- eli vaaka-akselilla yleensä aika-arvo ja y- eli pystyakselilla jokin tietoarvo, esimerkiksi sensorin mittaama lämpötila. (8.)

C#-ohjelmassa käytettäväksi graafiksi valittiin ilmainen netistä saatava luokkakirjasto nimeltä ZedGraph. ZedGraphilla voi piirtää esimerkiksi viiva-, piiras- ja pylväsdiagrammeja. ZedGraphissa on monenlaisia vapaavalintaisia säätömahdollisuuksia zoomauksesta väreihin, fontteihin ja akselien säätöihin,

mutta myös perusasetuksilla saa vaivattomasti piirrettyä hienoja kuvaajia (kuva 3). (9.)

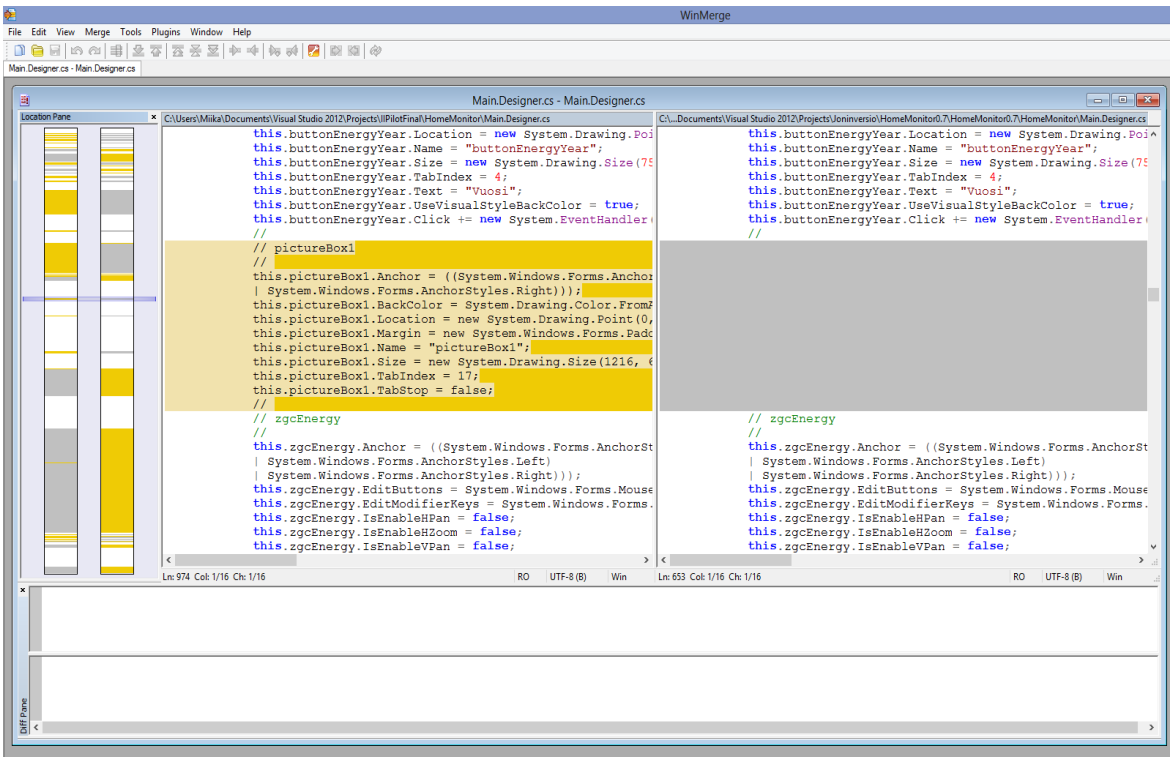


KUVA 3. ZedGraphillä toteutettu pylväsdiagrammiesimerkki (10)

2.2 WinMerge koodin integrointiin

WinMerge on avoimen lähdekoodin ilmainen vertailu- ja yhdistämisohjelma Windowsille. WinMergellä voidaan vertailla tekstimuotoisia tiedostoja helposti ymmärrettävästi ja hallitusti. Se on erityisen kätevä ohjelmakoodin integrointivaiheessa eli silloin, kun kahden eri ohjelmoijan koodimuutoksia halutaan yhdistää yhdeksi toimivaksi kokonaisuukdeksi. Tosin vain pienet muutokset voidaan integroida turvallisesti. (11.)

WinMergeen avataan kaksi vertailtavaa tiedostoa joko File-valikosta valitsemalla Open tai työkaluvalikosta keltaista kansiokuvaketta painamalla (kuva 4). WinMerge asettaa tiedostot vertailtavaksi vierekkäin samalle tasolle. Se näyttää uudet rivit keltaisella värillä ja puuttuvat rivit harmaalla värillä. Valkoinen väri tarkoittaa että kyseisten rivien välillä ei ole eroavaisuuksia.



KUVA 4. WinMergen käyttöliittymä. Vertailussa kaksi versiota MainDesigner.cs-tiedostosta.

Koodin yhdistely tapahtuu valitsemalla hiiren oikealla painikkeella joku värialue, joka kattaa yhden rivin tai useita rivejä. Alue voidaan siirtää joko vasemmalta oikealle tai oikealta vasemmalle. Pyrkimyksenä on saada tarpeelliset koodirivit molemmille puolille molemmista puolista identtiset eli valkoiset. Tällöin kummallekkin koodarille saadaan lopulta täsmälleen sama versio koodista. (12.)

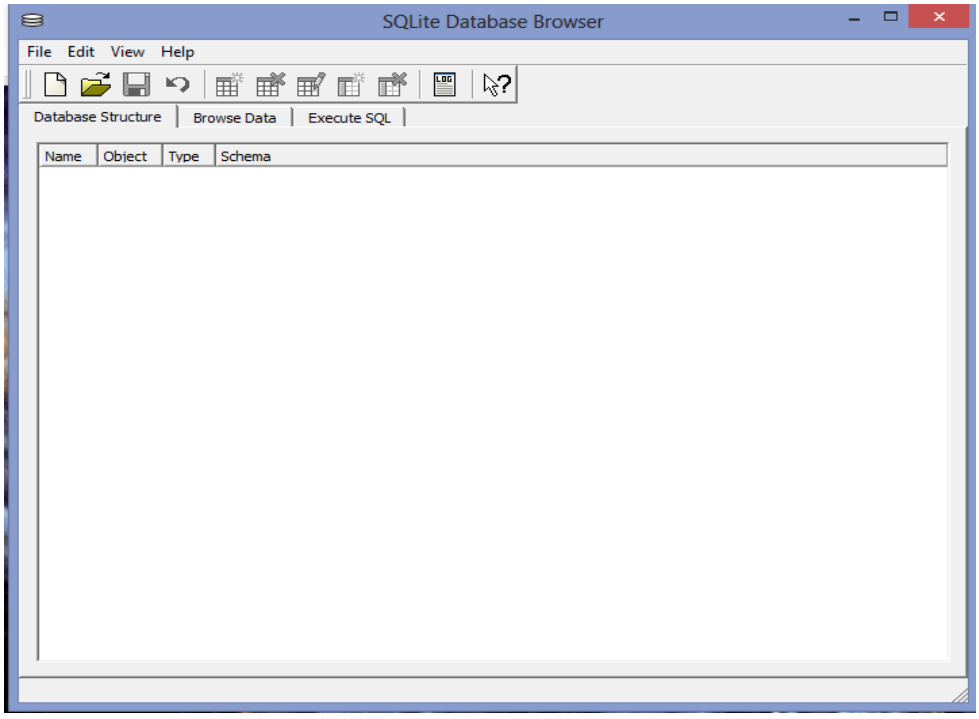
Yhdistelytoimintoa ei tule sekoittaa versionhallintaan, joka vertailee ja yhdistelee muutokset lähes automaattisesti ja pitää kirjaa tapahtumista. Ohjelmiston versiohallinta voidaan jakaa neljään päätoimenpiteeseen, jotka ovat versiointi, versioiden merkitseminen, versioiden välisten erojen tunnistaminen ja versioiden tallentaminen. Yleisimpiä versionhallintaohjelmistoja on Git, RCS, SVN, BitKeeper ja Microsoft Visual Source Safe. (13.)

WinMergellä ohjelmakoodin yhdistelystä on vastuussa käyttäjä itse. Jos yhdistelyssä sattuu virheitä, on niitä mahdoton perua WinMergen sulkemisen jälkeen. Tästä syystä kannattaakin aina ennen yhdistelyä ottaa tiedostoista varmuuskopiot talteen. Pääsääntönä on, että mitä vähemmän ohjelmoijat ovat tehneet samoihin tiedostoihin muutoksia, sitä helpompaa muutosten yhdistäminen WinMergellä on.

2.3 SQLite-tietokannan hallintatyökalu

SQLite-tietokannan hallintaan käytettiin työssä työkalua nimeltä **SQLite Database Browser**. Se on Qt-ohjelmointikielellä toteutettu yksinkertainen ja kevyt SQLite-tietokannan sisällön tarkasteluun ja muokkaukseen tarkoitettu ohjelma. Ohjelmaa ei tarvitse asentaa vaan se on latauksen jälkeen toimintavalmis. (14.)

SQLite Database Browserin käyttöliittymässä on aika itseselitteiset välilehdet Database Structure, Browse Data ja Execute SQL. Ensimmäisellä tarkastellaan SQLite-tietokannan rakennetta, toisella voidaan tarkastella taulujen sisältöjä tarkemmin ja kolmannella voi suorittaa SQL-käskyjä tekstimuodossa. Lisäksi ohjelmassa on muutamia perustoimintoja esimerkiksi uuden rivin lisäämiseen ja valitun rivin poistamiseen. (Kuva 5.)



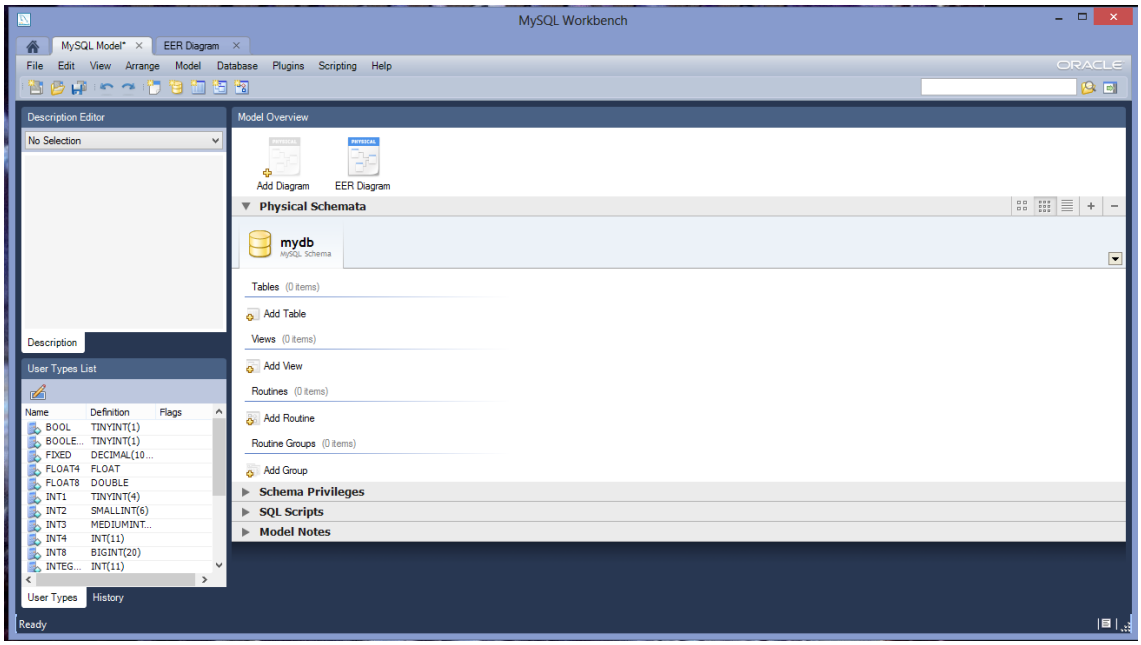
KUVA 5. SQLite Database Browserin käyttöliittymä

2.4 MySQL Workbench tietokannan mallintamiseen

Työssä käyttämäni **MySQL Workbench -**ohjelman versio 5.2.45 on virallinen, joskin ilmainen (saatavilla myös maksullinen versio) MySQL-tietokannan mallintamiseen ja hallitsemiseen tarkoitettu ohjelma, joka on nykyään Oraclen

omistuksessa. Sillä voi suunnitella ja toteuttaa taulujen rakenteet ja piirtää taulujen mallit viitesuhteineen. Sillä voi luoda MySQL-tietokannan suoraan SQL-lausein, mutta en käyttänyt tässä työssä ohjelmaa muuten kuin tietokannan mallinnukseen. MySQL-tietokantajärjestelmän SQL-kielen syntaksi on lähellä SQLiten vastaavaa, joten MySQL Workbench soveltui käytettäväksi tässä työssä. (15.)

Käyttöliittymässä näkyy vakiona vain MySQL Model -välilehti (Kuva 6). Siitä voidaan klikata Add Diagram jolloin saadaan toisena välilehtenä EER Diagram, jossa voidaan suorittaa varsinainen tietokannan mallinnus.



KUVA 6. MySQL Workbenchin käyttöliittymä

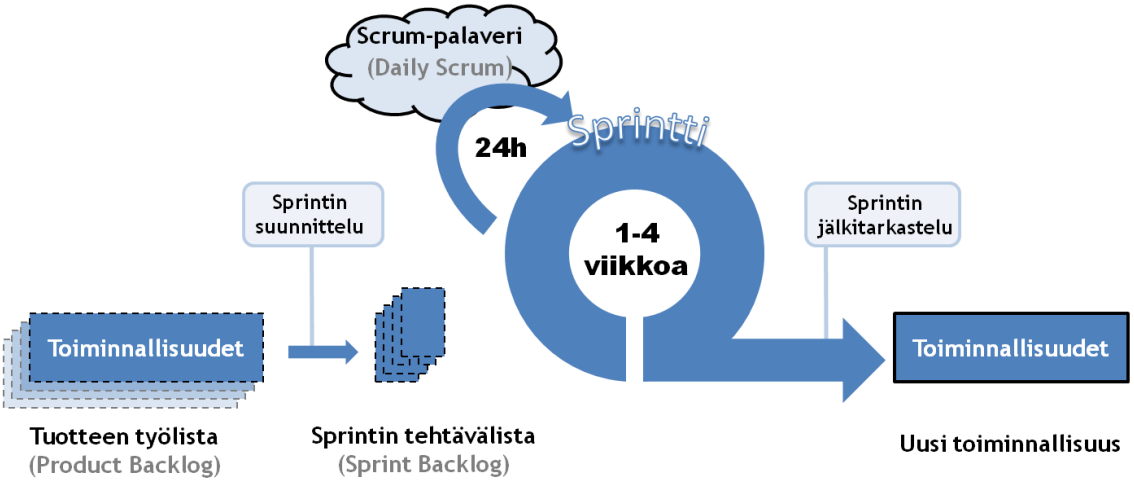
3 SCRUM-TUOTEKEHITYSMALLI

Scrum on 1990-luvun alusta asti käytetty ketterä ohjelmistokehityksen projektinhallintamalli. Projektinhallinta tarkoittaa yleisesti resurssien oragnisointia ja hallintaa sellaisella tavalla että projekti voidaan päättää suunnitellun sisältöisenä ja laatuisena aikataulun sekä budjetin mukaisesti. Käytettäviin resursseihin luetaan esimerkiksi raha, työvoima, raaka-aineet, energia, tila ja palkat. Resurssien lisäksi huomioidaan esimerkiksi viestintä, laatu ja riskit. (16.)

Scrumin avulla suunnitellaan työtehtävät, seurataan projektin edistymistä ja pyritään julkaisemaan eriasteisia julkaisuversioita ohjelmistosta. Scrum ei itsessään ole tekniikka eikä työkalu, vaan viitekehys jossa on tiettyjä sääntöjä. (17.)

3.1 Scrum-prosessi

Scrum-prosessi on esitetty kuvassa 7. Scrumissa listataan ohjelman toiminnallisuudet eli ominaisuudet Tuotteen työlista -nimiseen dokumenttipohjaan. Toiminnallisuuksia suunnitellaan maksimissaan kuukauden, mielellään vain parin viikon aikajaksoille, jotka ovat Scrumissa nimeltään sprinttejä.



KUVA 7. Scrum-prosessi (18)

Sprinteille luodaan tehtävälista, johon ominaisuudet on pilkottu pienemmiksi osasiksi eli tehtäviksi, joita voidaan kutsua myös taskeiksi (kuva 7). Scrum-kehitystiimi toteuttaa suunnitellut tehtävät. Tulosten saantia seurataan erilaisin palaverein. Sprintille annetaan maali, joka on enintään muutaman lauseen tiivistelmä tuloksista mitä kyseisessä sprintissä pitäisi saavuttaa. Koko projektille luodaan sopiva määrä sprinttejä sen keston mukaan. (19, s. 7.)

3.2 Scrumin ketteryys

Scrumin ketteryys tulee siitä että osaaminen jaetaan, työtuloksia tarkastellaan jatkuvasti ja riskit estetään jo ennen kuin niistä pääsee aiheutumaan suurempia vahinkoja. Riskit voidaan ennakoida kirjoittamalla riskilista. Scrumin on sanottu olevan helppo ymmärtää mutta vaikea hallita. (19, s. 3.)

Hyvä suunnitelmallisuus on myös koko ajan läsnä. Jos suunniteltu tehtävä jää sprintin päättyessä kesken, se voidaan siirtää seuraavaan sprinttiin tai poistaa kokonaan. Sprintin tehtävälistaa voidaan päivittää sprintin aikana, mutta kuitenkaan kesken sprintin ei voida poistaa siihen suunniteltuja tehtäviä kokonaan. (19, s. 3.)

3.3 Scrum-suunnitelma- ja seurantadokumentit

Scrum voidaan dokumentoida esimerkiksi Excel-taulukkoon, johon luetellaan kaikki tehtävät alakohtineen ja tiloineen. Asiakirjat luetaan Scrum-tuotoksiksi. (19, s. 11.)

Tässä työssä pidin kirjaa tehtävistä ja niihin käytetyistä tunneista Scrum-dokumentaatiossa johon kuului seuraavat sivut:

- Resurssien käytettävyys, josta ilmenee projektin yleinen aikataulutus arkipäivien, lomapäivien ja muihin projekteihin käytettyjen päivien määränä.

- Julkaisusuunnitelma, jossa kerrotaan projektin jäsenien määrä, sprinttien määrä sekä päivässä tehtävien tuntien määrä. Lisäksi siinä on jaettu jokaisen sprintin alku- ja loppupäivämäärät ja eritelty montako työpäivää kussakin sprintissä on. Julkaisuunnitelma siis jakaa projektin sprintteihin.

- Kaikki tuotteen ominaisuudet (engl. all product items), johon lisätään kaikkien sprinttien yksittäiset työtehtävät ja jota päivitetään sitä mukaa kun tehtäviä keksitään lisää tai muutetaan tai siirretään. Jokaisen tehtävän kohdalla voidaan arvioida työhön käytettävä täysien työpäivien määrä, tehtävän prioriteetti ja lisähuomioita. Tehtävät jaetaan prioriteetin mukaan erillisille alueille esimerkiksi erittäin korkeisiin, korkeisiin, keskitasoisiin ja mataliin tehtäviin. Prioriteetti kertoo mitkä tehtävät on saatava ensiksi valmiiksi.

- Tuotteen ominaisuuslista (engl. product backlog) eli tuotteen kehitysjono, josta ilmenee kaikkien sprinttien tehtävät eli taskit. Tehtävät erotellaan ID-luvuilla ja nimillä sekä niille annetaan tila ja mahdollisia lisähuomautuksia. Tiloja ovat yleisesti: ”Ei aloitettu”, ”Kesken”, ”Valmis” ja ”Hylätty”. Tuotteen ominaisuuslistaan lisätään sprintin tehtäviin käytetyt tuntimäärät aina kyseisen sprintin lopetuksessa. Nämä sprintin tunnit vähentävät sitten projektin kokonaistuntimäärää ja tuntien käyttöä voidaan samalla seurata.

- Sprintin tehtävälista (engl. sprint backlog), joita tulee yksi jokaista sprinttiä kohti. Jokaisessa sprintin tehtävälistassa on tehtäviä eli taskit on jaoteltu myös pienempiin osakokonaisuuksiin. Esimerkiksi tehtävässä ”Tietokannan suunnittelu” voisi olla osatehtävä ”Tietokannan speksaus”. Sprintin tehtävälistassa kirjataan kuka Scrum-kehittäjätiimin jäsen tehtävää on toteuttanut ja paljonko tunteja hän on tehtävään käyttänyt minäkin päivänä.

3.4 Scrum-tiimin roolit

Scrum-tiimi koostuu Scrum-mestarista (engl. scrum master), Scrum-kehitystiimistä (engl. development team) ja tuotteen omistajasta (engl. product owner). (19, s. 4.)

Scrum-mestari ohjaa Scrumin käyttöä ja johtaa Scrum-palavereja. Scrum-mestari ei kuitenkaan ohjaa yksin tehtävien tekoa vaan vastuu ja osaaminen jakautuu koko Ccrum-tiimin kesken. Scrum-mestari vastaa siitä, että kaikki

ymmärtävät ja käyttävät Scrumia. Scrum-mestarit tekevät tämän varmistamalla, että Scrum-tiimit pitäytyvät Scrumin teoriassa, käytännöissä ja säännöissä. Scrum-mestari on Scrum-tiimin palveleva johtaja. (19, s. 6.)

Kehitystiimin jäseniä ovat varsinaiset kehittäjät, jotka ovat työtehtävästä ja osaamisasteesta riippumatta samanarvoisia tiimin jäseniä. Kehitystiimi pitää itse huolen Scrumin läpiviennistä Scrum-mestarin johdolla. Kehitystiimin koko on oltava riittävän pieni jotta se pysyy ketteränä ja riittävän suuri jotta se saa valmiiksi merkittävän määrän työtä. (19, s. 5.)

Tuoteomistaja on vastuussa työn tulosten valmistumisesta ja tiimin jäsenten informoimisesta ja koko projektin kehityskaaren läpiviennistä. Hän ei sinällään puutu Scrum-viitekehyksen käyttöön projektissa. (19, s. 4.)

3.5 Scrumin palaverityypit

Scrumissa on erilaisia palaverityyppejä. Projektin alussa pidetään projektin aloituspalaveri, jossa sovitaan projektille aloitus- ja lopetusajat, sprintit, projektin tavoitteet ja dokumentaatioden jakelukanava. (19, s. 8.)

Ennen jokaista sprinttiä pidetään sprintin suunnittelupalaveri, jossa mietitään yleisesti mitä sprintissä tehdään, miten tehdään ja millä tehdään, sekä jaetaan tehtävät tiimin jäsenten kesken (19, s. 8).

Joka päivä pidetään päiväpalaveri (engl. daily Scrum), jossa tarkastellaan henkilö kerrallaan mitä kyseinen henkilö on tehnyt eilen, miten hän on tehtävässään onnistunut, onko ongelmia ilmennyt, miten ongelmat voidaan ratkaista ja mitä henkilö tulevana päivänä tulee tekemään. Päiväpalaveri pidetään Scrum-tiimin kesken seisten, jotta pysytään käsiteltävässä asiassa ja nähdään selkeästi kuka on puhevuorossa eikä palaveri veny liian pitkäksi. (19, s. 9.)

Sprintin lopussa pidetään sprinttikatselmus, jossa tarkastellaan kehitetty tuoteversio ja sopeutetaan tarvittaessa tuotteen kehitysjonoa. Sprintin katselmoinnin aikana Scrum-tiimi ja sidosryhmät selvittävät yhteistyössä, mitä sprintissä kehitettiin. Koko ryhmä pohtii, mitä voidaan ja kannattaa tehdä

seuraavaksi, jotta sprinttikatselmus antaa hyvän pohjan seuraavan sprintin suunnittelupalaverille. (19, s. 10.)

Sprintin retrospektiivi antaa Scrum-tiimille tilaisuuden tarkastella työskentelyään ja tehdä suunnitelman kehitysprosessin parannuksille, jotka toteutetaan seuraavassa sprintissä. Sprintin retrospektiivi pidetään sprinttikatselmuksen jälkeen ja ennen seuraavan sprintin suunnittelupalaveria. Palaveri rajataan enintään kolmeen tuntiin kuukauden sprintille. Lyhyemmille sprinteille varataan suhteessa vähemmän aikaa. (19, s. 11.)

4 TIETOKANNAT

4.1 Yleistä tietokannoista

Suurin osa kaikesta maailman tiedosta on jonkinlaisessa tietokannassa tallessa. Esimerkiksi kun Youtubesta etsitään videota hakukenttään annetulla sanalla, tekee käyttäjä tietämättään tietokantahaun. Tietokantoja käytetään kaikkialla: sähköpostipalveluissa, pankkitiedoissa, henkilötiedoissa, omistustiedoissa, kalentereissa, videopalveluissa, peleissä jne. Tietokanta on tietojärjestelmä, jossa on joukko yhteenkerättyä organisoitua tosielämästä peilautuvaa tietoa. Tieto voi olla itsenäistä tai toiseen tietoon liittyvää, ja tietoon voi päästä käsiksi yksi tai useampi käyttäjä. (20.)

Tietokannoissa tieto on yleensä taulukkomuodossa (engl. table) riveissä (engl. row) eli tietueissa ja sarakkeissa (engl. column) eli kentissä. Tietokannan tauluissa tieto voi olla tekstiä, numeerisia arvoja tai binääridataa eli kokonaisia tiedostoja. Ohjelmiston kannalta tietokantaa tarvitaan säilyttämään tietoa ohjelman suorituskertojen välillä (20.)

Tietokannan avulla voidaan säilyttää suurta määrää tietoa ja etsiä haluttu tieto tietyillä hakuehdoilla. Esimerkiksi taulukosta 1 voitaisiin etsiä tarpeellisia henkilön tietoja nimellä, osoitteella tai puhelinnumerolla, koska siinä on sarakkeet ID, Nimi, Osoite ja Puh. Tietokannan taulujen sarakkeet ovat ikäänkuin selityksiä joille annetaan luontivaiheessa vaihtumaton nimi, tietotyyppi ja maksimipituus.

TAULUKKO 1. Esimerkki yksinkertaisesta tietokantataulusta, jossa on henkilöiden tietoja

ID	Nimi	Osoite	Puh
1	Matti Meikeläinen	Nollakatu 0	555-1234456
2	Maija Meikeläinen	Nollakatu 0	555-11111111

Tietokantojen sarakkeiden tietotyyppejä ovat kokonaisluku integer, liukuluku float, double tai decimal, merkkijono varchar tai string, teksti text, päivämäärä-aika-yhdistelmä datetime ja binääridata blob joka voi olla esimerkiksi kuva- tai äänitiedosto. Lisäksi on tyyppi tyhjälle eli NULL. Sarakkeille annetaan luontivaiheessa määrite NULL tai NOT NULL. Määritys kertoo saako kyseiseen sarakkeeseen antaa tyhjää tietoa vai ei. (21.)

Jokaisessa taulussa yksi sarake on aina perusavain eli pääavain (engl. primary key), joka tarkoittaa riviä yksilöivää kenttää eli saraketta. Yksilöinti mahdollistaa tarkan tiedonhaun. Perusavaimen saraketta kutsutaan yleisesti ID:ksi (ID = Identification = tunnistus). Yleensä perusavain on tyyppiä integer auto_increment eli kasvava kokonaisluku joka kasvaa aina yhden suuremmaksi kuin tauluun viimeksi lisätyn ID-rivi:n luku. Ohjelmallisesti ID-kenttään annetaan arvo NULL riviä lisättäessä, mutta auto_increment-ominaisuus laskee itsestään luvun. Jokaisen ID-kentän on oltava uniikki. (22.)

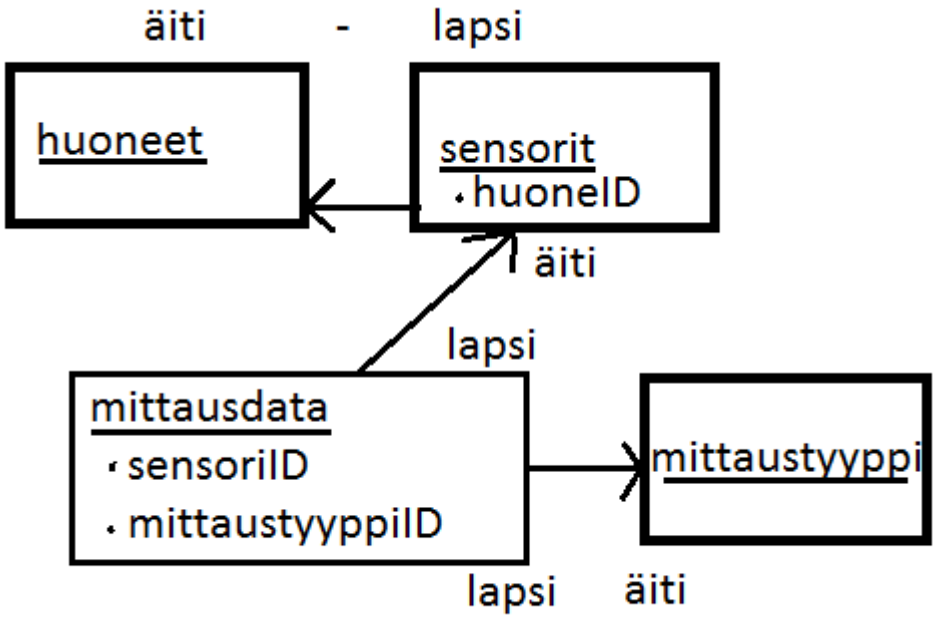
4.2 Relaatiotietokanta

Tietoa voidaan hallita tietokantajärjestelmän avulla. Yleisimpiä tietokantajärjestelmiä ovat maksulliset Microsoft SQL Server, Microsoft Office Access, Microsoft Visual FoxPro ja Oracle ja ilmaiset MySQL, PostreSQL, SQLite ja Firebird. Microsoft SQL Serveristä on myös ilmainen Compact Edition (CE) -versio. Luetelluista jokainen sisältää erilaisia ominaisuuksia, mutta ne pohjautuvat kuitenkin SQL-tietokantakieleen ja ovat relaatiotietokantoja. (23.)

Relaatiotietokanta on nykyisin yleisimmin käytetty tietokantamuoto. Muita tietokantamuotoja ovat oliotietokanta ja ns. No-SQL. Relaatiot eli suhteet muodostuvat taulujen välille muodostaen äiti-lapsitauluja. Lapsitaulun jokaisella rivillä on viittaus äititaulun ID-kenttään eli sillä tunnistetaan mihin äititaulun-riviin lapsitaulun rivi perustuu. Relaatiotietokannan perussääntönä on että äidillä voi olla useita lapsia, mutta lapsilla ei useita äitejä. (24.)

Esimerkiksi sensorien tiedot ovat yhdessä taulussa, sensorien mittaustulokset toisessa taulussa ja huoneet, joihin sensorit liittyvät, kolmannessa taulussa.

Taulut muodostavat keskenään suhteita. Lapsitaulun jokaisella rivillä on viittaus äititaulun ID-kenttään, paitsi jos NULL-arvoja on mahdollista antaa kyseisille sarakkeille jolloin kenttä ei viittaa mihinkään äititaulun riviin. (Kuva 8.)



KUVA 8. Esimerkki tietokannan äiti-lapsi-taulurakenteesta

Kun kaksi taulua on relaatiossa, on molemmilla tauluilla omat ID:t rivien tunnistusta varten sekä lapsitaulussa sarake joka sisältää viittauksen äititaulun ID-kenttään. Tätä saraketta kutsutaan viiteavaimeksi (engl. foreign key). (25.)

Relaatio voi toteutua kolmella eri tyypillä: yhden suhde yhteen, yhden suhde moneen ja monen suhde moneen. Näistä tavoitteellisin tyyppi on yhden suhde moneen.

Esimerkki yhden suhde yhteen -tilanteesta voisi olla että yhdellä käyttäjällä on vain yksi salasana eli yhdellä salasanalla vain yksi käyttäjä. Tällöin olisi kaksi taulua: käyttäjät-taulu, ja salasanat-taulu. Yhden suhde yhteen -suhteet kannattaa kuitenkin yhdistää samaan tauluun. Käyttäjätaulussa voi olla käyttäjän nimi ja salasana eikä erillistä salasana-taulua tarvita.

Monen suhde moneen tarkoittaa esimerkiksi tilannetta, jossa henkilöllä on useita asuntoja ja tällä asunnolla on useita omistajia. Tällöin kannattaa luoda

kolmas taulu henkilöt-taulun ja asunnot-taulun välille. Taulussa Henkilön_asunto on viittaus sekä henkilöt-taulun tiettyyn ID:hen että asunnot-taulun tiettyyn ID: hen.

Yhden suhde moneen tarkoittaa esimerkiksi, että yhdessä koulussa on monta opiskelijaa mutta yhdellä opiskelijalla ei ole montaa koulua. Toisena esimerkkinä yhteen huoneeseen liittyy useita sensoreita, mutta jokainen sensori liittyy vain yhteen huoneeseen. Tämä ajatus on pohjana tämänkin työn tietokantasuunnittelulle.

4.3 SQL-tietokantakieli

SQL (Structured Query Language) on IBM:n kehittämä standardoitu tietokantakieli, joka on kehitetty erityisesti relaatiotietokantoja varten. Uusin ISO-standardi on vuodelta 2008. Kaikki yleiset relaatiotietokannat käyttävät kielenään SQL-tietokantakieltä. (26.)

SQL:n perusoperaatiot ovat tietokantakyselyjä (engl. query). Kyselyt toteutetaan varattujen sanojen avulla. Yleisimpiä SQL-kyselyjä on tiedon hakeminen, tiedon päivittäminen, tiedon lisääminen ja tiedon poistaminen (kuva 9). Lisäksi kyselyiksi luetaan taulujen ja indeksien luonnit, muuttamiset ja poistamiset ja muutamat muut käskyt. (26.)

```
SELECT id, tietue FROM taulu WHERE quux = 'xyzzy' ORDER BY id DESC;
UPDATE taulu SET kentta = 'esimerkki' WHERE id = 42;
INSERT INTO taulu (kentta,toinenkentta) VALUES ('tietoa', 5);
DELETE FROM taulu WHERE kentta = 123;
```

KUVA 9. Yleisimpiä SQL-kyselyjä (26)

4.3.1 Tietueiden hakeminen tietokannasta

Tietueiden eli rivejen hakeminen alkaa varatulla sanalla SELECT (kuva 10), jonka jälkeen on kerrottava haettavien sarakkeiden nimet, esimerkiksi ”id” ja ”tietue”. Vaihtoehtoisesti voidaan valita kaikki sarakkeet merkillä ”*”. Sen jälkeen hakulauseeseen on lisättävä varattu sana FROM jonka jälkeen kerrotaan taulun nimi eli mistä tietokannan taulusta tieto haetaan. Tämän jälkeen tulee ei-pakollinen osio: hakuehdot.

```
SELECT id, tietue FROM taulu WHERE quux = 'xyzzy' ORDER BY id DESC;
```

KUVA 10. SELECT-lause-esimerkki (26)

Hakuehto tulee varatun sanan WHERE jälkeen. Esimerkiksi valitaan valitusta taulusta vain tietueet joissa sarakkeessa nimeltä quux lukee merkkijono ’xyzzy’. Hakuehdon jälkeen syntaksissa seuraavana tulee järjestelysanat ORDER BY ja näiden jälkeen jonkun sarakkeen nimi. ORDER BY siis järjestää löydetyt hakutulokset jonkin sarakkeen mukaan. Ilman ORDER BY:ta löydetyt hakutulokset järjestyvät ID-kentän mukaan.

4.3.2 Tietueen lisääminen

Tietueita voidaan lisätä INSERT INTO sanojen avulla (kuva 11). Sanojen jälkeen on annettava taulun nimi ja sulkuihin tulee kenttien nimet. Sulkujen jälkeen tulee varattu sana VALUES. Sen jälkeen tulee sulkuihin oikeantyyppisinä annettavat tiedot. Annettu tieto ’tietoa’ on merkkijono, joten kentta on merkkijono eli varchar-tyyppiä, kun taas numero 5 viittaa toinenkentta-sarakkeeseen, joten tämä sarake toinenkentta on kokonaisluku eli integer.

```
INSERT INTO taulu (kentta,toinenkentta) VALUES ('tietoa', 5);
```

KUVA 11. INSERT INTO -lause-esimerkki (26)

4.3.3 Tietueen päivittäminen

Jo olemassaolevia tietueita voidaan päivittää UPDATE- ja SET-sanojen avulla SQL-kyselyllä (kuva 12). Näiden sanojen väliin tulee taulun nimi. SET-sanan jälkeen tulee päivitettävän sarakkeen nimi ja WHERE-ehto rajaa päivitettävien sarakkeiden määrää. Tässä tapauksessa päivitetään sarakkeeseen, jonka id on 42 ja nimi on kentta, merkkijono ’esimerkki’. Päivittäminen ei siis ota kantaa siihen mitä kentässä on ennen lukenut. Päivitettävän tiedon tyypin on vastattava sarakkeen tyyppiä tai SQL antaa virheilmoituksen eikä päivitys onnistu.

```
UPDATE taulu SET kentta = 'esimerkki' WHERE id = 42;
```

KUVA 12. UPDATE SET -lause-esimerkki (26)

4.3.4 Tietueen poistaminen

Tietueiden poistaminen tapahtuu sanalla DELETE, jonka jälkeen annetaan sana FROM, jonka jälkeen seuraa taulun nimi (kuva 13). Tämän jälkeen tulee vapaavalintainen WHERE-ehto joka rajaa poistettavien tietueiden määrää. WHERE sanan jälkeen annetaan sarakkeen nimi ja arvo. Tässä tapauksessa poistetaan taulusta vain tietue jonka kentta-sarakkeen arvona on 123. Ilman WHERE-ehtoa tästä taulusta poistetaan kaikki tietueet.

```
DELETE FROM taulu WHERE kentta = 123;
```

Kuva 13. SELECT-lause-esimerkki (26)

4.3.5 Keskeisimmät SQL-funktiot

SQL-tietokantakielessä on monia valmiita funktiota (27). Suurin arvo voidaan etsiä MAX-funktion ja pienin arvo MIN-funktion avulla. Keskiarvo saadaan AVG-funktiolla. Kaikki nämä funktiot pitää sisällyttää haku- eli select-lauseeseen ja lisäksi niihin pitää lisätä hakulauseen loppuun sanat GROUP BY jos halutaan ryhmitellä tuloksia jonkin sarakkeen mukaan. Muiden funktioiden selitykset jäävät tämän työn ulkopuolelle, koska niitä ei käytetty.

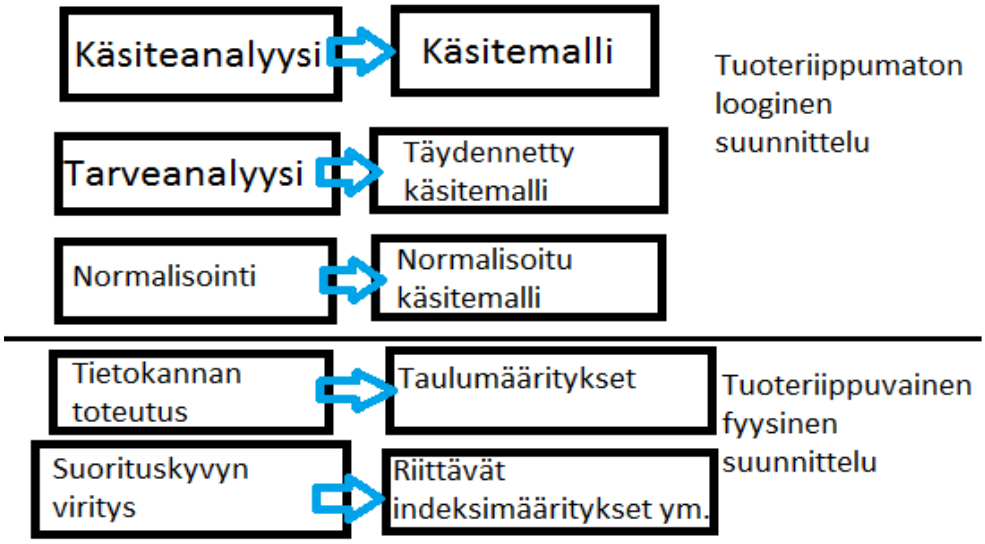
5 TIETOKANNAN SUUNNITTELU

5.1 Yleistä tietokantasuunnittelusta

Tietokantasuunnittelun tärkeimpänä pääperiaatteena on tiedon toiston välttäminen. Toistuva tieto vie turhaa tilaa, ylläpito muodostuu hankalaksi koska kaikki kopiot on myös päivitettävä ja ylläpito-operaatioilla voi olla odottamattomia sivuvaikutuksia. Turhaa tietoa, jota ei koskaan käytetä mihinkään, on turha myöskään lisätä tietokantaan. (28.)

Tietokannan suunnitteluputki

Tietokannan suunnittelu voidaan ajatella suunnitteluputkeksi, jossa on monta vaihetta (kuva 14). Aluksi tehdään käsiteanalyysi, joka tuottaa käsitemallin. Sen jälkeen tehdään tarveanalyysi, joka tuottaa täydennetyn käsitemallin. Tämän jälkeen tietokantaa normalisoidaan, jolloin saadaan normalisoitu käsitemalli. Normalisoinnin jälkeen seuraa tietokannan toteutusvaihe joka tuottaa varsinaisen tietokannan taulumääritykset. Toteutusvaiheen alussa voidaan luoda tietokannalle prototyyppi. Viimeinen vaihe on suorituskyvyn viritys, jolloin suorituskykyä pyritään maksimoimaan. Suunnitteluputken vaiheita tehdään toistuvasti ja yhä tarkentaen tietokannan määrittelyä, suunnittelua ja toteutusta.



KUVA 14. Suunnitteluputki (29, s.24)

Käsiteanalyysistä käsitemalliksi

Tietokannan suunnitteluputki eli -prosessi alkaa käsiteanalyysillä jossa mietitään, mitä tietoa tietokantaan on tarpeellista sijoittaa ja mitä näistä tiedoista tullaan käyttämään usein ja mitä harvemmin. Tällöin ei vielä tarvitse vielä miettiä tietokannan teknistä toteutusta, suoritusasioita eikä fyysisiä rakenteita. Käsiteanalyysi tuottaa yleisluontoisen kuvauksen tietokannasta, joka voidaan mallintaa karkeaksi käsitemalliksi. Se on käytännössä ER-kaavio. Käsiteanalyysissä ei paneuduta siihen miten tietokanta tullaan toteuttamaan, vaan mitä siihen tulee. (29, s.24.)

ER-kaavio

Tietokannan mallintaminen tarkoittaa ER-kaavion piirtämistä. ER tulee sanoista Entity-Relationship eli käytännössä se tarkoittaa taulujen suhteita toisiinsa. Suhteet voidaan esittää nuolilla taulujen välillä, ja erilaiset nuolet tarkoittavat erilaisia suhteita, esimerkiksi yhden suhde yhteen tai yhden suhde moneen. ER-kaaviosta ilmenee myös tietokannan taulut ja tauluissa esiintyvien sarakkeiden määrä, tyypit sekä nimet. (30.)

Tarveanalyysi

Suunnitteluputken toinen vaihe on tarveanalyysi. Tietotarveanalyysissä käsitemallia tarkennetaan sovelluksen perusteella ja lisätään puuttuvat tiedot jolloin tuloksena saadaan täydennetty käsitemalli. Analyysiin kuuluu myös mm. lajittelukriteerien ja tieto-, käyttäjä- ja tapatumamäärien selvitys. Näitä tietoja tarvitaan mm. testausta varten ja tieto vaikuttaa myös tietokannan hallintajärjestelmän valintaan. (*29*, s.24.)

5.2 Tietokantasuunnittelun tärkeitä sääntöjä

Hyvän tietokannan suunnittelu alkaa siitä, että tuntee hyvät säännöt tietokannan rakenteelle. Alle olen koonnut Tietokantojen suunnittelu ja indeksointi -kirjasta (*29*) löytämiäni tärkeimpiä sääntöjä.

Relaatiotietokannoissa pitää olla vahva tietoriippumattomuus, mikä tarkoittaa että uusia tauluja ja sarakkeita pitää voida helposti lisätä ilman vaikutusta nykyohjelman toimintaan. Uusia indeksejä voidaan luoda ja vanhoja poistaa tai muuttaa ilman muutosta sovelluksen toimintaan. Relaatioiden pitää olla kunnossa, jotta tietojen haku helpottuu ja nopeutuu. (*29*, s. 10–12.)

Jokaisella taululla on oltava uniikki perusavain. Rivit eli tietueet tunnistetaan perusavaimen avulla yksilöllisiksi. Perusavain ei saa olla NULL eli on oltava avaineheys. (*29, s.9.*)

Tieto pitää normalisoida eli lajitella sopivan pieniin osiin. Esimerkiksi etu- ja sukunimi ovat omissa kentissään, jolloin voidaan etsiä tietoa sekä henkilön etunimellä ja sukunimellä erikseen. Tieto on tällöin atomista. Poikkeuksena aikaleimat voivat olla tyyppiä datetime, jossa on päivämäärä ja aika samassa kentässä. (*29*, s.25.)

Eheyden on oltava kunnossa. Tietojen ja taulujen on oltava oikeanlaisia, yhdenmukaisia, ristiriidattomia ja samaa tuoreustasoa. Esimerkiksi asiakkaalla ei saa olla tietokannassa kahta tietuetta joissa kummassakin lukee osoite eri lailla. Siinä tielanteessa ei voida tietää kumpi osoitteista on oikea ja kumpi ei. (*29*, s.11.)

Arvoalue-eheyden on oltava kunnossa. Kenttään annetaan vain oikeanmuotoinen arvotyyppi. Esimerkiksi int-tyyppistä tietoa ei saa laittaa float-tyyppiseen kenttään eikä päivämäärää saa sijoittaa väärässä formaatissa datetime-kenttään. (*29*, s.11.) Tosin SQLitessä tämä on vähän tulkinnanvaraista.

Viite-eheyden on oltava kunnossa. Viite-eheys estää tyhjän arvon laittamisia, varmistaa, että viittaukset ovat oikeellisia, ja estää tietueen poiston, jos siihen on viittauksia muualla. Äiti-taulusta ei voi poistaa riviä, jos lapsitaulussa on viittaus äiti-taulun johonkin kenttään, sillä muuten lapsi jäisi orvoksi. (*29*, s.11.)

Tietokannan optimointi eli suorituskyky pitää ottaa huomioon tietokantaa suunniteltaessa. Rakenne on pidettävä suhteellisen yksinkertaisena, koska

hakuihin tulee tällöin vähemmän taululiitoksia. Suorituskykyä voidaan virittää asetuksilla ja indeksien käytöllä. (29, s.21–22.)

Lisäksi omia huomioitani ovat muun muassa seuraavat:

Taulujen ja rakenteiden on oltava täysin valmiita ennen kuin yhtään riviä tietoa sijoitetaan tauluun. Esimerkiksi minkään kentän sisällön ei pidä perustua toisen taulun kenttään.

Syöttörajoitteet on mietittävä hyvin. Esimerkiksi puhelinnumero-kentälle on turhaa varata yli 20 merkin pituista kenttää. Tosin SQLitessä tästäkään ei tarvitse kantaa huolta.

5.3 Normalisointi

Tietokannan normalisointi on tärkeä osa suunnittelua. Normalisointi jaetaan useimmiten kolmeen (tai neljään) vaiheeseen. Normalisoinnin tavoite on saada toimivampi tietokantarakenne. Normalisoidussa tietokannassa tietojen toistaminen on minimoitu, tietokannan rakenne on tehokas päivitysten kannalta, rakenne on helpompi pitää yhdenmukaisena sillä tiedot tarvitsee päivittää vain yhteen paikkaan ja rakenne on muutosjoustava. Käytännössä tämä tarkoittaa taulujen rakenteen uudelleen järjestelyä siten, että normalisointivaiheiden ehdot toteutuvat. (29, s. 86.)

5.3.1 Ensimmäinen normaalimuoto

Tietojen on oltava atomisia eli ”poista toistuvat ryhmät ja moniarvoiset sarakkeet”. Tämä toteutetaan niin, että moniarvoiset tiedot on poistettava ja siirrettävä omiin tauluihinsa. Taulurakenne on suunniteltava muutosjoustavaksi. Seuraavaksi kirjasta poimittu esimerkki kuinka huonosta taulurakenteesta tehdään hyvä normalisoimalla se ensimmäiseen normaalimuotoon. (29, s. 88.)

Esimerkkitaulussa (taulukko 2) näytetään henkilön henkilötunnus, suku- ja etunimi ja nykyinenpalkka. Käyttäjät saattavat olla tälläiseen tietokantatauluun tyytyväisiä. (29, s. 87.)

TAULUKKO 2. Henkilo-taulu (29, s. 87–89)

Henkilo			
htun	sukunimi	etunimi	palkka
3343	Järvi	Ritva	2400
2245	Joki	Pekka	2300

Jos haluttaisiin säilyttää tietokannassa useampi palkkatieto tietylle henkilölle, ei nykyinen taulurakenne soveltuisi siihen. Taulua pitäisi muokata taulukon 3 kaltaiseksi kahta palkkatietoa varten. (29, s. 87)

TAULUKKO 3. Henkilo-taulua muokattu niin, että siihen voidaan sijoittaa kahden palkan tiedot. (29, s. 87–89)

Henkilo						
htun	sukunimi	etunimi	pvm1	palkka1	pvm2	palkka2
3343	Järvi	Ritva	1.1.2000	2400	1.1.2001	3000
2245	Joki	Pekka	1.1.2000	2300	1.1.2001	2200

Sarakkeet pvm ja palkka muodostavat toistuvan ryhmän. Jos kuitenkin halutaan säilyttää yhä enemmän paikkatietoja, olisi taulukkoon tehtävä aina muutoksia eli lisättävä aina vain uusia sarakkeita, mikä ei ole järkevää. Jos tällöin ei ole vanhoja palkkoja, tauluun jää tyhjiä sarakkeita, mikä ei sekään ole hyvä asia. Ohjelmallisesti ratkaisu olisi typerä, koska hakulauseeseen pitäisi kirjoittaa kaikkien haettavien sarakkeiden nimet. (29, s. 87.)

Seuraavaksi voidaan kokeilla ratkaisua, jossa on moniarvoinen sarake palkkahistorialle (taulukko 4). Voidaan todeta, että palkkahistorian ylläpito (lisäykset, muutokset ja poisto) sekä laskutoimitusten tekeminen

palkkahistoriasarakkeesta tulisivat hyvin ongelmallisiksi. Moniarvoisesta sarakkeesta pitäisi osata pilkkoa tieto erikseen eri muuttujiin ohjelmakoodissa.

TAULUKKO 4. Henkilo-tauluun lisätty moniarvoinen sarake palkkahistorialle (29, s. 87–89)

Henkilo				
htun	sukunimi	etunimi	palkka	palkkahistoria
3343	Järvi	Ritva	3000	2000, 2400
2245	Joki	Pekka	2200	2000, 2300

Ratkaisu on poistaa toistuvat ryhmät ja moniarvoiset sarakkeet eli jakaa nykyinen Henkilo-taulu kahteen eri tauluun (taulukot 5 ja 6) ensimmäisen normaalimuodon mukaisesti. Palkka-taulussa on viittaus Henkilo-taulun htun-sarakkeeseen eli jokaiselle henkilölle voidaan luoda useita rivejä palkkatietoja jotka sisältävät päivämäärän ja palkkaluvun. (29, s. 88.)

TAULUKKO 5. Henkilo-taulu, muutettu takaisin alkuperäiseksi (29, s. 87–89)

Henkilo			
htun	sukunimi	etunimi	palkka
3343	Järvi	Ritva	3000
2245	Joki	Pekka	2300

TAULUKKO 6. Palkka-taulu (29, s. 87–89)

Palkka		
htun	pvm	palkka
2245	1.1.2000	2400
2245	1.1.2001	2010
3343	1.1.2000	2300
3343	1.1.2001	2055

Henkilo-taulusta voitaisiin jättää palkka-sarake kokonaan pois, koska se on nyt myös palkka-taulussa. Huomattiin että normalisointi toteutuu käytännössä pilkkomalla tauluja useaan eri tauluun, jotka sisältävät tiettyjä kokonaisuuksia: henkilöjen tiedot Henkilo-taulussa ja näiden henkilöiden palkkatiedot Palkka-taulussa.

5.3.2 Toinen normaalimuoto

Toiseen normaalimuotoon liittyy vahvasti käsite funktionaalinen riippuvuus. Voidaan sanoa, että sarake A on funktionaalisesti riippuvainen sarakkeesta B, jos jokaista B:tä kohti on aina korkeintaan yksi A:n arvo. Tälläinen riippuvuus on siis taulujen sarakkeiden välillä. Esimerkiksi sarake sukunimi on funktionaalisesti riippuvainen sarakkeesta htun koska tiettyä sukunimeä kohti on useita eri henkilötunnuksia. Toisin päin riippuvuutta ei ole, joten htun ei ole riippuvainen riippuvainen sukunimestä. (29, s. 90.)

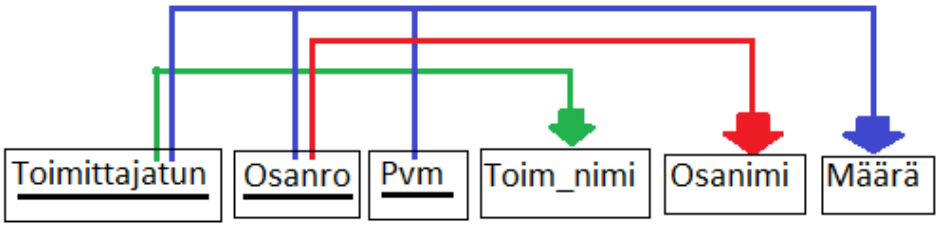
Alla oleva esimerkkitaulu (taulukko 7) kuvaa auton varaosien tilauksia. Taulussa on moniosainen perusavain. Rivit tunnistetaan tietyn toimittajan sekä tietyn osan tilauksen tiettynä päivänä tehdyn tilauksen mukaan. Perusavainrivit on alleviivattu.

TAULUKKO 7. Auton varaosien tilauksia sisältävä taulu (29, s. 91)

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

Toisen normaalimuodon pääsääntö kuuluu: jos taulussa on moniosainen perusavain, kaikkien sarakkeiden tulee olla funktionaalisesti riippuvia koko perusavaimesta eikä siis vain osa-avaimesta. Jos jokaisen taulun avaimet koostuvat vain yhdestä attribuutista, ovat ne käytännössä heti toisen normaalimuodon täyttäviä. (29, s. 91.)

Yllä olevan taulukko 7:n funktionaaliset riippuvuudet on kuvattu kuvassa 15. Taulukko rikkoo toisen normaalimuodon sääntöä, koska kaikki sarakkeet eivät ole funktionaalisesti riippuvaisia perusavaimesta. Esimerkiksi Osanimi on riippuvainen vain Osanro-kentästä. (29, s. 91.)



KUVA 15. Sarakkeiden riippuvuudet auton varaosien tilauksia sisältävälle taululle (*29*, s. 91.)

Asia voidaan korjata normalisoimalla toiseen muotoon jakamalla kentät eri tauluihin (taulukot 8, 9 ja 10).

TAULUKKO 8. Toimittajan tiedot (29, s. 92)

Toimittajatun	Toim_nimi
1202	Varaosa Oy
1275	A-osat Oy

TAULUKKO 9. Osien tiedot *(29, s. 92)*

Osanro	Osanimi
707b	Tulppa 12
708b	Tulppa 13

TAULUKKO 10. Toimittajat ja osat yhdistävä taulu, jossa on lisäksi Pvm- ja Määrä-sarakkeet (29, s. 92)

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

Alkuperäisessä ratkaisussa (taulukko 7) oli seuraavat heikkoudet:

1. Toimittajia tai osia ei voi tallentaa etukäteen erikseen.
2. Poistettaessa vanhoja tilauksia saattaa jonkin toimittajan tai osan nimi kokonaan hävitä.
3. Toimittajien ja osien nimet toistuvat turhaan moneen kertaan.

4. Toimittajan tai osan nimen muuttuessa koko taulu on käytävä läpi ja tehtävä runsaasti päivityksiä, mikä on hidasta.

Normalisoinnin jälkeen heikkoudet ovat poistuneet. Voidaan etukäteen tallentaa sadoittain toimittajia toimittaja-tauluun. Voiidaan myös poistaa vanhoja tilauksia poistamatta toimittajaa. Toimittajan nimen muutos päivitetään ainoastaan yhdelle riville tosi nopeasti. Sama koskee osien tietoja. (29, s. 92.)

5.3.3 Kolmas normaalimuoto

Kolmannen normaalimuodon pääsääntö on seuraavanlainen. Jokaisen sarakkeen pitää olla funktionaalisesti riippuvainen vain perusavaimesta eikä siis mistään tavallisesta sarakkeesta. Esimerkkitaulu (taulukko 11) toteuttaa toisen normaalimuodon, koska taulussa on yksiosainen perusavain ja muut sarakkeet ovat funktionaalisesti riippuvaisia htun-sarakkeesta.

TAULUKKO 11. Henkilöiden postiosoitteita (*29*, s. 92.)

htun	sukunimi	etunimi	katuosoite	postinro	postitoimipaikka
2245	Joki	Pekka	Klaarantie 4	00200	HELSINKI
2475	Ranta	Eeva	Isokaari 4b2	00200	HELSINKI

Kuitenkin huomataan, että postitoimipaikka on funktionaalisesti riippuvainen myös postinro-sarakkeesta, koska jokaiseen postinumeroon liittyy vain yksi postitoimipaikka. Lisäksi taulussa on turhaa toistoa. Ratkaisu on purkaa taulu kahteen eri tauluun (taulukot 12 ja 13).

TAULUKKO 12. Henkilöiden postiosoitteita ilman postitoimipaikkaa (*29*, s. 92.)

htun	sukunimi	etunimi	katuosoite	postinro
2245	Joki	Pekka	Klaarantie 4	00200
2475	Ranta	Eeva	Isokaari 4b2	00200

TAULUKKO 13. Postinumero määrää postitoimipaikan (*29*, s. 92.)

Postinro	Postitoimipaikka
00200	HELSINKI
...	...

6 SOPIVAN TIETOKANNAN VALINTA SENSORIDATOILLE

6.1 Tietokannan valintaperusteet

Tietokannan valintaa varten keräsin listan ominaisuuksia eli kriteereitä, joiden pitäisi toteutua jotta kyseinen tietokanta valittaisiin tuotteen ohjelmistoon. Näitä olivat seuraavat:

- *Ilmainen ja avoin lähdekoodi.* Valittavan tietokannan ei pitäisi aiheuttaa suuria rahallisia kuluja firmalle.

- *Skaalautuva.* Sitä mukaa, kun järjestelmään lisätään erilaisia sensorityyppejä ja lisää sensoreita, täytyy tietokannan skaalautua sen mukaan suuremmaksi. Tämän pitää tapahtua ilman, että tietokannan rakenteeseen enää kosketaan.

- *Helppokäyttöinen ja vähän huoltoa vaativa.* Mitä vähemmän huoltamista ja muuttamista tietokanta vaatii tulevaisuudessa, sen parempi.

- *Paikallinen*. Tietokannan pitää aluksi toimia lokaalisti yhden käyttäjän koneella ilman palvelinta. Palvelimen ylläpitäminen maksaa aikaa ja vaivaa.

- *Kompakti*. Tietokannan koon pitää pysyä mahdollisimman pienenä eli se ei saa viedä paljoa tilaa käyttäjän pc:ltä.

- *Nopeat hakuoperaatiot*. Tietokannan pitää pystyä hakemaan suuresta määrästä dataa esimerkiksi tietyn aikavälin historiatiedot tietylle sensorille joka on tietyssä huoneessa. Jos tietokannan hakuoperaatio on hidas, joutuu käyttäjä odottelemaan, mikä ei ole koskaan hyvä asia.

- *Yhteensopivuus C#-kielen kanssa*. Tietokantaa pitää pystyä käsittelemään suoraan C#-kielellä ohjelmakoodissa.

- *Helppo asennettavuus tai ei ollenkaan asennusta*. Tietokannan pitää tulla tuotteen mukana ja alkaa toimia loppukäyttäjän tietokoneella ilman vaikeata asennusprosessia ja mieluiten ilman, että käyttäjän tarvitsee

tehdä yhtään mitään. Vain esimerkiksi uusien tilojen ja sensorien lisääminen järjestelmään jää käyttäjän tehtäväksi. Asennus on tehtävä mahdollisimman automaattiseksi.

- *Hyvä vaihdettavuus.* Tietokanta pitää olla vaihdettavissa jatkossa esimerkiksi monen käyttäjän palvelinpohjaiseen tietokantajärjestelmään.

Näiden tietojen perusteella päädyin valitsemaan SQLite-tietokannan. Se täyttää kaikki annetut kriteerit lähes täydellisesti. Lisäksi kokeilin vertailun vuoksi MySQL- ja Microsoft SQL Server CE -tietokantoja.

6.2 SQLite

SQLite on ilmainen, vapaan lähdekoodin SQL-relaatiotietokantajärjestelmä, joka on toteutettu C-kielellä alle 250 kilotavun kokoiseksi paketiksi. Sen yksi pääkehittäjä on Dwayne Richard Hipp. SQLitestä saa muokata itselleen sopivia versioita täysin vapaasti, ja näin useat kehitystiimit ovatkin tehneet Hippin kannustamina. SQLite on suunniteltu kevyeksi, pieneksi ja erityisesti sulautettujen järjestelmien ohjelmistosovelluksiin sopivaksi. (31.)

Koko SQLite-tietokanta on yhden sqlite-päätteisen tiedoston sisässä kovalevyllä perinteiseen flat-file-tyyliin eikä, kuten tietokantajärjestelmät yleensä, palvelin-asiakas (engl. server-client) -tyyppisesti. Tiedostoja voidaan luoda rajattomasti ja yhdistää ATTACH-komennolla. Kun tietokanta luodaan, siihen tulee automaattisesti taulu nimeltä sqlite_sequence. Se on päätaulu, jossa on tietoa muiden taulujen nimistä ja rivimääristä. (32.)

Vaikka haluttaisiin käyttää perinteistä palvelin-asiakastietokantaa, on SQLite kehitys- ja testausvaiheessa hyvä vaihtoehto, ja siitä voidaan siirtyä aika vaivattomasti isompaan ja mahdollisesti kaupalliseen tietokantaan. Yksittäinen tietokanta on helppo kopioida, koska riittää että kopioi yhden tiedoston. SQLite-tiedostopääte on cross-platform, joten se toimii periaatteessa kaikissa ympäristöissä riippumatta käyttöjärjestelmästä. Lähteenä olevalla Youtube-videolla pääkehittäjä Dwayne Richard Hipp kertoo SQLiten tärkeimmistä ominaisuuksista (33).

Kun tietokantaan kirjoitetaan tai sieltä luetaan tietoa, SQLite-tiedosto on tapahtuman aikana lukossa, koska halutaan olla varmoja, että tapahtuma on oikeasti mennyt kovalevylle asti. Ilman tätä sisäistä varmistusta tietoa saattaisi hävitä jos laitteelta katkeaa sähkövirta. Virran katkeaminen on yleistä sulautettujen järjestelmien laitteissa. Jos dataa häviäisi tiedon eheys kärsisi ja tietokanta saattaisi korruptoitua peruuttamattomasti.
Tietokannanlukitusominaisuus ei ole yleistä muissa tietokantajärjestelmissä vaan yleensä vain yksi taulu lukitaan tapahtuman ajaksi. SQLite ei sovellu hyvin usean käyttäjän yhtäaikaiseen käyttöön lukituksen vuoksi, mutta paikallisessa standalone-ohjelmassa ei tule ongelmia jos ohjelmakoodissa on otettu lukitus huomioon. (32.)

Luku- ja kirjoitusoperaatioiden maksiminopeus on sidottu kovalevyn pyörimisnopeuteen (32.) Esimerkiksi kovalevyllä jonka pyörimisnopeus on 7200 pyörähdystä minuutissa maksimi siirto-operaatioiden (engl. transaction) määrä on noin 60. Siirrot eli transaktiot ovat ison käskyjoukon toteuttamista varten luotavia tapahtumia. Siirtoihin voidaan sisällyttää tietokannan tehokkuuden optimoinnin mukaan esimerkiksi 45 000 käskyä sekunnissa.

SQLite tulkitsee sen taulujen sarakkeiden tyypit erikoisesti verrattuna normaaleihin SQL-tietokantoihin (32.) Se tulkitsee esimerkiksi kaikki merkkijonot, tekstit ja vastaavat tyypit text-tyyppiseksi ja kaikenlaiset numeeriset arvot real-tyyppisiksi. Sarakkeille ei myöskään ole tarpeen antaa maksimipituutta taulua luotaessa.

SQLite varaa tilaa vain sen verran muistista kuin oikeasti on tarpeen eikä tyyppi varchar(100) esimerkiksi varaa 100 merkille tilaa ellei kenttään sijoita oikeasti 100:aa merkkiä. Jos sijoitetaan vaikka 26 merkkiä, tilaa varataan vain 26 merkin verran. Kuitenkin tässä työssä käytin taulujen luonnissa juurikin esimerkiksi varchar(45)- enkä text-tyyppiä, jotta tietokannan taulujen luonnit ovat tulevaisuudessa suoraan MySQL-tietokannan kanssa yhteensopivia.

6.3 MySQL

MySQL on yleinen ja ilmainen vapaan lähdekoodin relaatiotietokantajärjestelmä joka on perustettu vuonna 1995. MySQL käyttää SQL-tietokantakielen ISO-

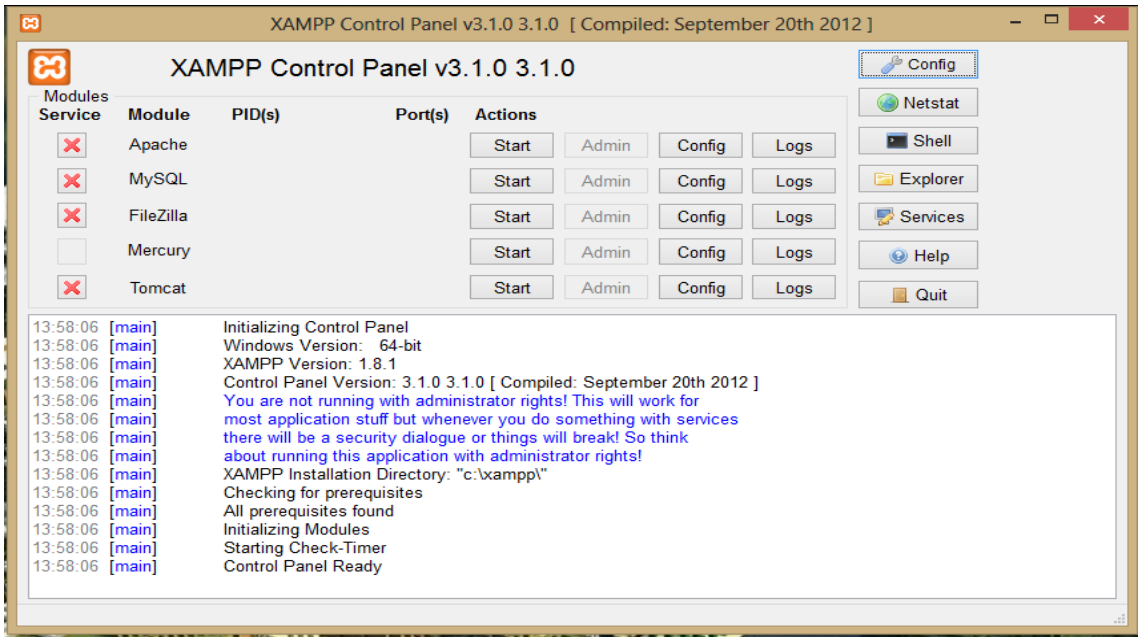
standardia. MySQL:ää kehittää ruotsalainen yritys MySQL AB ja se on siirtynyt vuosien saatossa Sun Microsystemsin omistuksesta Oraclen omistukseen. MySQL:stä on tosin saatavilla myös maksullinen versio jos GNU-lisenssiä ei haluta käyttää. (32.)

MySQL on tehokas tietokantahauissa, sillä on paljon valmiita funktioita ja mahdollisuus käyttää viiteavaimia, indeksejä, näkymiä ja liipaisimia (engl. trigger). MySQL on hyvin suosittu erilaisissa web- eli verkkopalveluissa. Siksi se vaatii toimiakseen MySQL-palvelimen, jota ajetaan esimerkiksi LAMP-, WAMP- tai XAMPP-palvelimen yhtenä osana. (34.)

XAMPP-palvelin

XAMPPin nimi tulee sanoista Crossplatform Apache MySQL PHP/Perl. Se tarkoittaa, että XAMPP toimii sekä Linux- ,Windows-, Mac- että Solaris-ympäristöissä. LAMP toimii ainoastaan Linuxilla ja WAMP ainoastaan Windowsilla. (35.)

XAMPP on helppo asentaa ja laittaa päälle. Asennus hoituu normaalisti installerista ja päälle laittaminen XAMPP Control Panel -ohjelmiston kautta. Kun MySQL halutaan toimimaan, laitetaan Apache- ja MySQL-palvelimet päälle Start-painikkeista. Config-painikkeista pääsee tarkastelemaan ja muuttelemaan asetustiedostoja. (kuva 16.)



KUVA 16. XAMPP-palvelimen ohjauspaneeli

Tietokanta haluttiin sisällyttää tässä työssä pääsovellukseen ilman kummempia asennuksia tai palvelimen ylläpitoa, joten siksi valittiin mieluummin SQLite.

6.4 Microsoft SQL Server CE

Microsoft SQL Server CE (Compact Edition) on ilmainen ja kompakti tietokantajärjestelmä joka on erinomainen vaihtoehto sulautettuihin järjestelmiin työpöytä- ja web-aplikaatioihin. SQL Server Compact 4.0 antaa kehittäjille yleisen muista SQL Server -tuotteista tutun ohjelmointirajapinnan ja relaatiotietokannan toiminnallisuuden, vahvan tiedon varastoinnin, käskyjen optimoinnin sekä luotettavan ja skaalautuvan yhteydenpidon. Microsoft SQL Server CE ei käytännössä vaadi hallintatoimenpiteitä ja se on tarkoitettu yhden käyttäjän työasemasovelluksiin. Tämä tietokantajärjestelmä tulee automaattisesti Visual Studio 2012:n mukana. (36.)

7 TIETOKANNAN TOTEUTUS SCRUMIN AVULLA

Jaoin opinnäytetyön toteutusosan Scrumin mukaisesti sprintteihin, joita loin neljä kappaletta. Ensimmäinen sprint oli nimeltään Sprint 0, koska se ei liittynyt suoraan pääohjelman eikä tietokannan koodaukseen vaan se oli niin sanottu perehdyttämisjakso. Sprintissä 1 suunnittelin ja aloin toteuttaa SQLite-tietokantaa, Sprintissä 2 pääpaino siirtyi datan hakemiseen tietokannasta SQL-lausein, pääohjelman käyttöliittymän toteuttamiseen ja tiedon näyttämiseen käyttöliittymässä graafilla. Sprintissä 3 jatkoin graafien parantelua ja uusimpien tietojen hakua tietokannasta sekä tein yleisesti käyttöliittymän ja tietokannan yhteistoimintaa.

En esittele tässä työssä pääohjelman käyttöliittymää, koska tuote ei ollut silloin vielä julkaisukelpoisessa tilassa sen osalta ja oli siten salaiseksi luokiteltu. Käyttöliittymän tekemiseen kului tässä työssä kuitenkin suuri määrä tunteja, joten siksi se on maininnan arvoista.

7.1 Resurssien käytettävyys

Projektin eli opinnäytetyön toteutusosalle suunnittelin Scrumin mukaisen aikataulutuksen (kuva 17). Toteutusosalle suunnittelin alku- ja loppupäivämäärän, projektin tekoon käytettävät päivät, jotka lasketaan siten, että arkipäivistä vähennetään lomapäivät ja muut päivät eli koulunkäynti, ja tunnit. Kokonaistuntimäärä toteutukselle oli 40 * 7 = 280, koska suunnittelin tekeväni työtä 7 tuntia päivässä 40 päivän ajan. Koko opinnäytetyön pituus oli 400 tuntia, josta toteutusosaan kuului 280 tuntia, joten tekstiosuuden kirjoittamiseen jäi 120 tuntia. Työn päätteeksi huomasin pysyneeni hyvin aikataulussa ja olin saanut toteutetuksi suunnittelemiani tehtäviä.

YRITYKSEN NIMI	Oy PremiSense Ltd
PROJEKTIN NIMI	Opinnäytetyö: tietokannan määrittely, suunnittelu ja toteutusta sensoridatalle
SCRUM MASTER	Jussi Vepsäläinen
PROJEKTIN AIKATAULU	10.12.2012 – 22.2.2013 + kirjoituosa 25.2.2013 - 29.3.2013

DOKUMENTIN NIMI	scrum_seuranta_miika_kontio_UUSI.xls
TAULUKON NIMI	Resurssien käytettävyys

koulu

Henkilö	ARKIPÄIVIÄ PROJEKTIN AIKANA	MUUT PROJEKTIT	LOMAT	MUUT	PROJEKTITYÖN TEKEMISEEN PÄIVIÄ	PROJEKTIYÖN TEKEMISEEN TUNTEJA
Miika Kontio	58,0	0,0	10,0	8,0	40,0	280,0
					0,0	0,0
					0,0	0,0
					0,0	0,0
Yhteensä	**58,0**	**0,0**	**10,0**	**8,0**	**40,0**	**280,0**

KUVA 17. Scrum-dokumentaation Resurssien käytettävyys

7.2 Julkaisusuunnitelma

Scrum-dokumentaation toisella sivulla eli Julkaisusuunnitelmassa (kuva 18) käsitellään toteutusosan sprinttien resurssit tarkemmin kuin Resurssien käytettävyydessä. Dokumentaatiosta ilmenee koko projektin ja jokaisen sprintin viikot, päivät ja tunnit sekä sprinttien uusien ominaisuuksien määrä. Sprintissä 2 oli todellisuudessa pari ominaisuutta lisää, jotka oli siirretty sprintistä 1 sinne.

PROJEKTIN NIMI	Opinnäytetyö: tietokannan määrittely, suunnittelu ja toteutusta sensoridatalle			
SCRUM MASTER	Jussi Vepsäläinen			
PROJEKTIN AIKATAULU	10.12.2012 – 22.2.2013 + kirjoituosa 25.2.2013 - 29.3.2013			
DOKUMENTIN NIMI	scrum_seuranta_miika_kontio_UUSI.xls			
TAULUKON NIMI	Julkaisusuunnitelma			
PROJEKTISSA JÄSENIÄ	1			
SPRINTTEJÄ PROJEKTISSA	5			
TUNTEJA PER PÄIVÄ	7			
PROJEKTIN RESURSSIT	10.12.2012 – 22.2.2013			
HENKILÖT	**KÄYTETTÄVISSÄ VIIKKOJA**	**KÄYTETTÄVISSÄ PÄIVIÄ**	**KÄYTETTÄVISSÄ TUNTEJA**	
Miika Kontio	15	40	280	
YHTEENSÄ		40	280	
SPRINTIN 0 RESURSSIT	10.12.2012 – 4.1.2012			
HENKILÖ	**VIIKKOJA SPRINTISSÄ**	**PÄIVIÄ SPRINTISSÄ**	**TUNTEJA SPRINTISSÄ**	**OMINAISUUKSIA SPRINTISSÄ**
Miika Kontio	4	12	84	6
YHTEENSÄ		12	84	6
SPRINTIN 0 MAALI:	*Aloituspalaveri, eri tietokantoihin tutustuminen, tietokannan valinta, perehtyminen, testiapplikaatioiden luominen tietokantojen testailuun, projektin www-sivut, dokumentaatiot*			
SPRINTIN 1 RESURSSIT	07.01.2013 – 25.1.2013			
HENKILÖ	**VIIKKOJA SPRINTISSÄ**	**PÄIVIÄ SPRINTISSÄ**	**TUNTEJA SPRINTISSÄ**	**OMINAISUUKSIA SPRINTISSÄ**
Miika Kontio	3	12	84	8
YHTEENSÄ		12	84	8
SPRINTIN 1 MAALI:	*Tietokannan vaatimuslista kirjotettuna, tietokannalla jonkinlainen mallikaavio valmis, huollon tietokanta toteutettuna, sqlite integroituna pääohjelmaan ja perusfunktiot kunnossa*			
SPRINTIN 2 RESURSSIT	28.1.2013 – 8.2.2013			
HENKILÖ	**VIIKKOJA SPRINTISSÄ**	**PÄIVIÄ SPRINTISSÄ**	**TUNTEJA SPRINTISSÄ**	**OMINAISUUKSIA SPRINTISSÄ**
Miika Kontio	2	8	56	2
YHTEENSÄ		8	56	2
SPRINTIN 2 MAALI:	*Sensoridataa alustavasti tietokannassa, sensoridatan haku aikaleimalla pääohjelmalle toiminnassa, tietokannan testaus*			
SPRINTIN 3 RESURSSIT	11.2.2013 – 22.2.2013			
HENKILÖ	**VIIKKOJA SPRINTISSÄ**	**PÄIVIÄ SPRINTISSÄ**	**TUNTEJA SPRINTISSÄ**	**OMINAISUUKSIA SPRINTISSÄ**
Miika Kontio	3	8	56	3
YHTEENSÄ		8	56	3
SPRINTIN 3 MAALI:	*Valmis hyvin pääohjelman kanssa toimiva tietokanta, toimivat käyrien piirrot graafeissa*			

Resurssien_käytettävyys | **Julkaisusuunnitelma** | Kaikki tuotteen ominaisuudet | Tuotteen ominaisuuslista | Sprintin 0 tehtävälista | Sprintin 1 tehtävälista | Spri

KUVA 18. Scrum-dokumentaation Julkaisusuunnitelma

7.3 Sprint 0

Sprint 0 aloitettiin projektin aloituspalaverilla, jossa tehtiin lähtötietomuistio (liite 1). Sprintin 0 ominaisuuksiin kuului hyvän tietokannan valinta ja alustava suunnittelu, testisovellusten tekeminen MySQL-, SQLite- ja Microsoft SQL Server CE -tietokannoille, Visual Studion oman C# Chart -graafin testaaminen ja muut asiat joita olivat mm. ohjelmistoasennus, palaverit ja Scrum-dokumentaation kirjoittaminen (taulukko 14). Sprintille annettiin taulukon 15 mukainen maali. Koko Scrum-dokumentaatio löytyy liittestä 8.

TAULUKKO 14. Tuotteen ominaisuuslista Scrumin Sprintissä 0

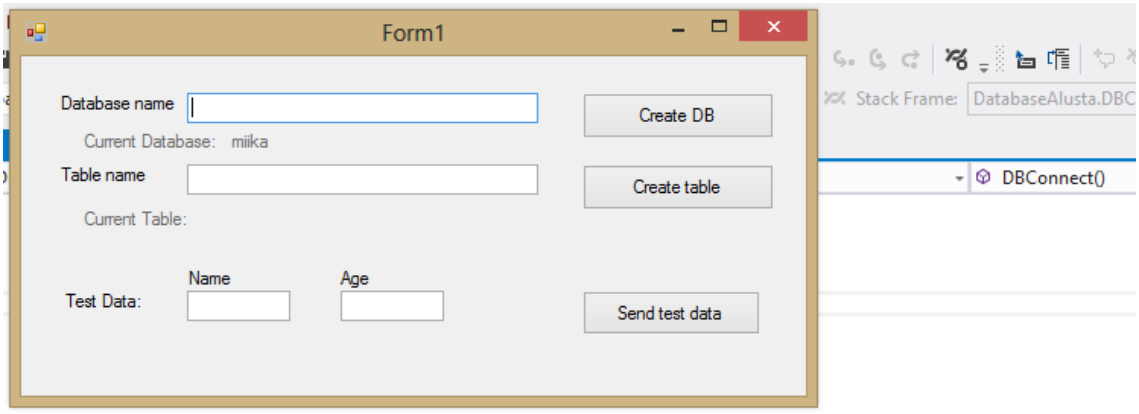
Ominaisuuden ID	Ominaisuuden kuvaus
1	MySQL C# ohjelma
2	SQLite C# ohjelma
3	Tietokannan valintakriteereihin tutustuminen ja tietokannan valinta
4	Tietokannan suunnittelu
5	C# Chart-graafi
6	Microsoft SQL Server CE C# ohjelma
	Muut asiat

TAULUKKO 15. Sprintin 0 maali

	SPRINTIN 0 MAALI: Aloituspalaveri, eri tietokantoihin tutustuminen, tietokannan valinta testiapplikaatioiden luominen tietokantojen testailuun, projektin www-sivut, dokumentaatiot

7.3.1 C#-testisovellus MySQL-tietokannalle

Osasin ennestään käyttää vain MySQL-tietokantaa eli tunsin sen SQL-syntaksin ja useimmat sen säännöistä. Lisäksi ainakin teoriassa tiesin kuinka sitä käytetään C#-ohjelmassa, joten se oli ensimmäisenä vaihtoehtona tietokannaksi. En panostanut testisovelluksen ulkonäköön tai moniin ominaisuuksiin. Sovellus ainoastaan ottaa MySQL-tietokantajärjestelmään yhteyden, ja sillä voi luoda tyhjiä tietokantoja ja valmiiksi määrittelemiäni tietokantatauluja sekä lähettää luotuun tauluun string-tyyppisenä Name-sarakkeeseen nimen ja int-tyyppisenä Age-sarakkeeseen iän ja siten lisätä uusia rivejä tietokantaan. (Kuva 19.)




```
private void CreateDatabaseConnection(string database)
{
    try
    {
        string connStr = "Server=localhost;port=3306;Database=" + database + ";Uid=root;Pwd=mysli;";
        connection = new MySqlConnection(connStr);
        System.Diagnostics.Debug.WriteLine("Connection succesfull to MySQL db");
    }
    catch (MySqlException ex)
    {
        MessageBox.Show(ex.ToString());
    }
}

public void CreateDatabase(string newDatabaseName)
{
    dataBaseName = newDatabaseName;

    this.OpenConnection();

    using (var cmd = connection.CreateCommand())
    {
        cmd.CommandText = "CREATE DATABASE IF NOT EXISTS " + dataBaseName + ";";
        cmd.ExecuteNonQuery();
    }

    this.CloseConnection();
    this.CreateDatabaseConnection(dataBaseName);
}
```

KUVA 19. C#-MySQL-testisovelluksen käyttöliittymä, ja ohjelmakoodia, jolla luodaan tietokanta

MySQL:n heikkous tähän tuotteeseen kuitenkin on se, että se vaatii palvelimen toimiakseen. Palvelimen ylläpitäminen vaatii tehokkaan palvelintietokoneen jota pitää pitää jatkuvasti päällä ja huoltaa säännöllisesti huoltokatkosten aikana. Toinen vaihtoehto olisi ostaa WebHosting-palvelua muualta ja pitää MySQL-palvelinta siellä palveluntarjoajan pc-koneella.

Kokeilin kuitenkin Sprint 0:n aikana XAMPP-palvelinta lokaalisti eli koneella, jossa työskentelin. Pidin Apache- ja MySQL-palvelimia päällä IP-osoitteessa 127.0.0.1, joka on sama kuin localhost. Latasin XAMPP:n osoitteesta http://www.apachefriends.org/en/xampp-windows.html ja asensin sen

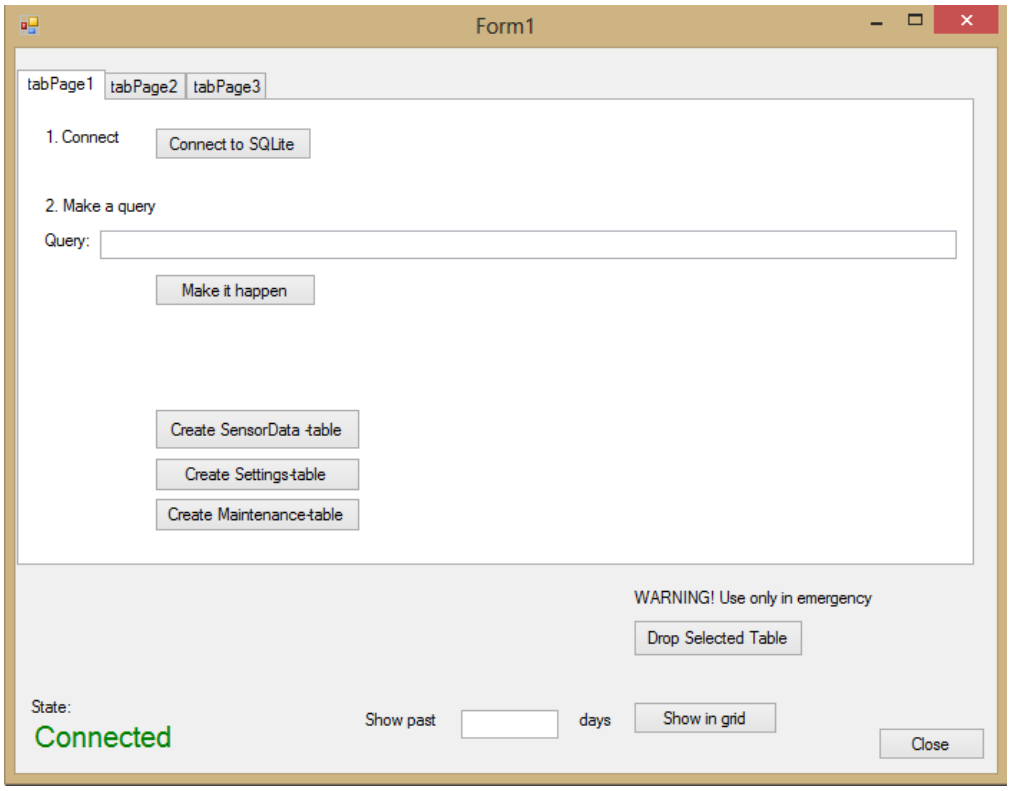
koneelleni. Vaihdoin XAMPP:n asetustiedostosta (config.php) käyttäjänimeksi root, salasanaksi mysli ja IP-osoitteeksi 127.0.0.1. Käytin näitä sitten C#-koodissa.

Huomasin, että tarvitsen MySQL-dll-kirjaston nimeltä MySQL Database Connector/NET 6.5.5. Latasin sen osoitteesta http://dev.mysql.com/downloads/connector/net/6.5.html, asensin sen ja otin sen C#-koodissa käyttöön lisäämällä DLL-kirjaston referenssiksi projektiin. Visual Studiossa valitaan Solution Explorerista References-kohdasta hiiren oikealla näppäimellä Add Reference. Tämän jälkeen etsitään Assemblies-listalta haluttu paketti MySql.Data. Lopuksi painetaan OK. Tämä DLL-kirjasto otetaan koodissa käyttöön tiedoston alkuun kirjoitettulla rivillä: using Mysql.Data.MySqlClient. Tämä ohje toimii hyvin myös myöhemmin esiteltävien SQLiten ja ZedGraphin DLL:ien lisäämisissä.

7.3.2 C#-testisovellus SQLite-tietokannalle

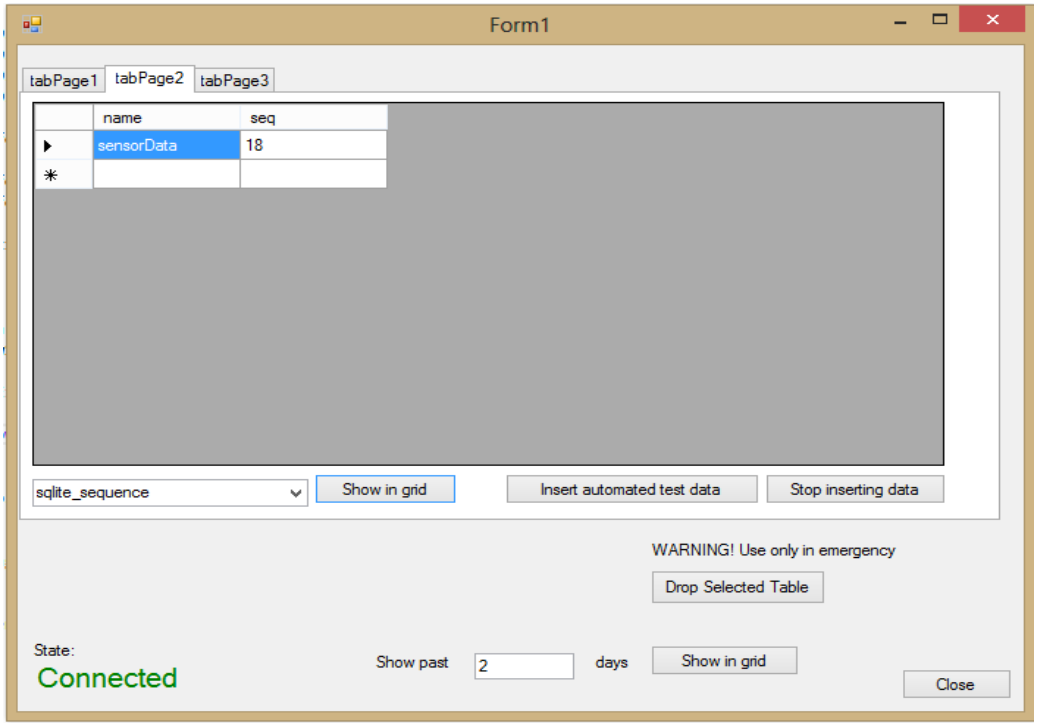
Seuraavaksi tein C#-testisovelluksen SQLitelle. Tein tämän sovelluksen jotta saisin SQLiten perustoiminnot käyttöön C#-ohjelmassa. Panostin paljon tämän sovelluksen toimintoihin ja koodiin, jotta voisin käyttää niitä myöhemmin Sprintissä 1 oikeassa pääsovelluksessa.

Tein testiohjelmaan kolme välilehteä. Ensimmäisellä välilehdellä (kuva 20) voi ottaa Connect to SQLite -painiketta painamalla yhteyden SQLite-tiedostoon, lähettää kokonaisia kyselyjä query-tekstikenttään kirjoittamalla ja Make it happen -painiketta painamalla tai luoda Create -painikkeilla valmiiksi koodissa määrittämiäni tauluja tietokantaan. Kuvan alaosassa on Drop Selected Table -painike, jota painamalla voidaan poistaa välilehdellä 2 valittu tietokantataulu. Show In Grid -painikkeella voidaan hakea Show past -tekstin jälkeiseen tekstikenttään annetun numeroarvon verran päiviä taaksepäin. Tämän aikavälihaun tulokset näytetään välilehdellä 2.



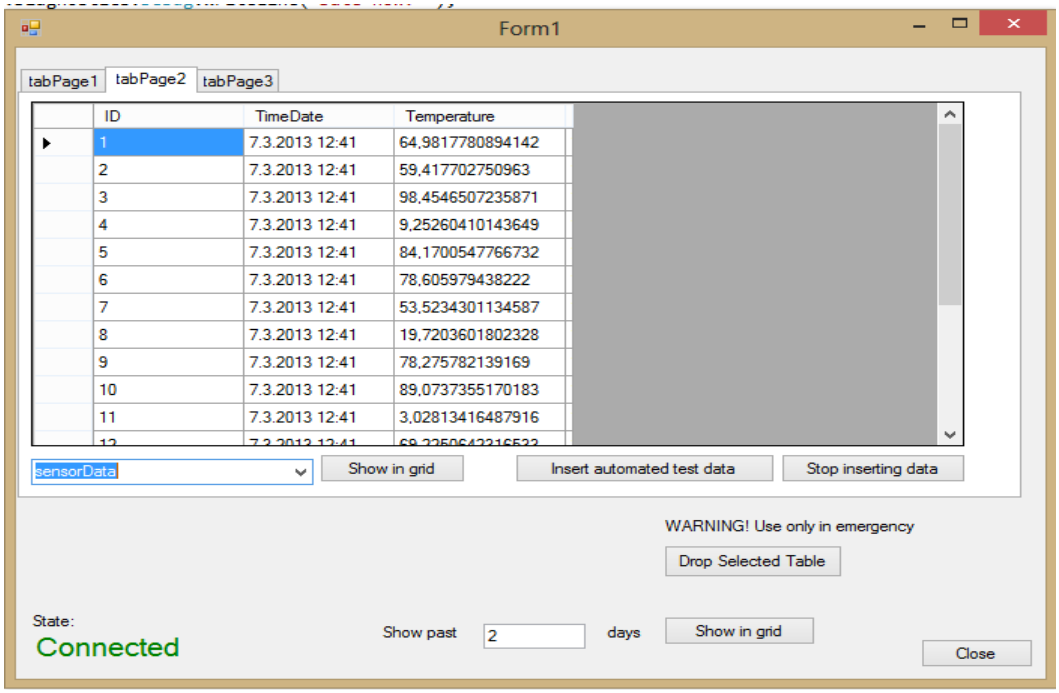
KUVA 20. C#-SQLite-testisovelluksen 1. välilehti

Toisella välilehdellä on iso harmaa alue johon voi hakea taulukkona alasvetovalikosta valitun tietokantataulun sisältöä painamalla Show in grid -painiketta. Alasvetovalikkoon haetaan SQLite-tietokannan kaikki taulut. Kuvassa 21 näkyy alasvetovalikossa ns. master-taulu sqlite_sequence. Tämän taulun ollessa valittuna Show in grid -painike näyttää taulukossa, että sensorData-tietokantatauluun on lisätty 18 kappaletta rivejä.



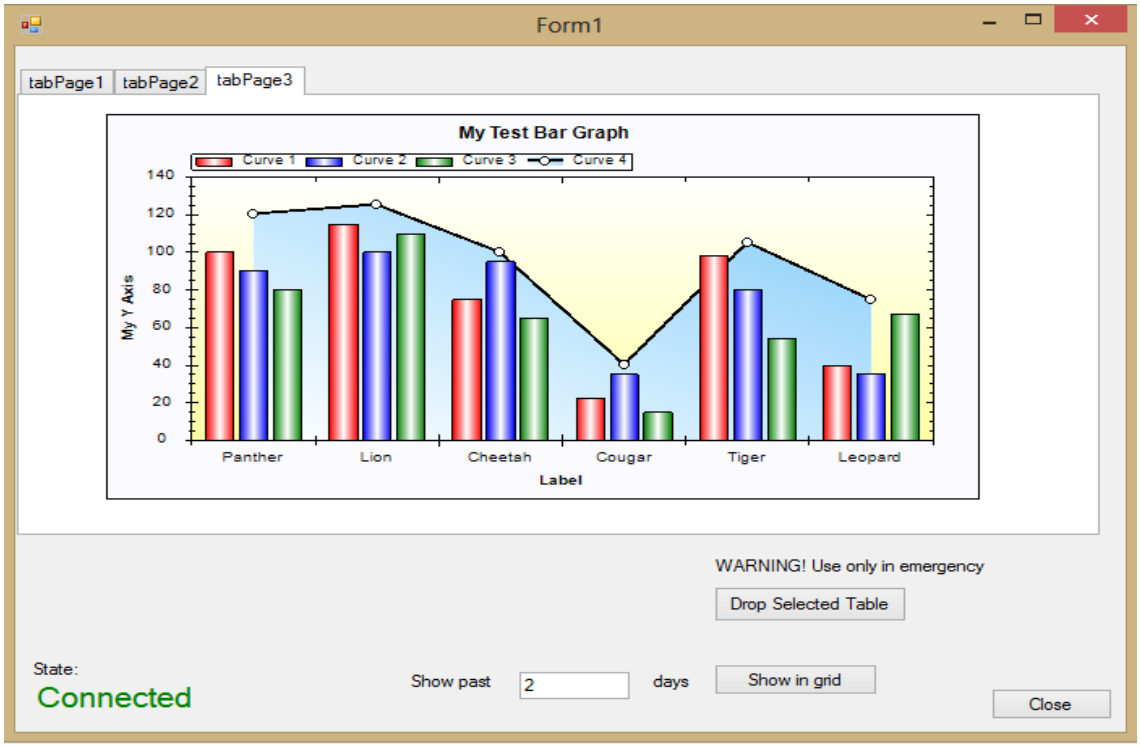
KUVA 21. C#-SQLite-testisovelluksen 2. välilehdellä sqlite_sequence-taulun sisällön tarkastelua

Lisäksi toisella välilehdellä voi valita sensorData-tietokantataulun ja painaa Insert automated test data -painiketta. Tällöin pyörähtää käyntiin säie, joka lisää nopeasti yksittäisiä rivejä tietokantaan, ja ne näytetään DataGridView -taulukossa (kuva 22). Stop inserting data -painike pysäyttää säikeen.



KUVA 22. SensorData-taulun sisällön tarkastelua C#-SQLite-testisovelluksen 2. välilehdellä

Kolmannella välilehdellä (kuva 23) on ainoastaan testin vuoksi piirretty ZedGraph-graafi. ZedGraphista kerrotaan lisää myöhemmin.



KUVA 23. ZedGraph C#-SQLite-testisovelluksen 3. välilehdellä

SQLiten asennus C#-ohjelmaan

SQLiteä ei tarvitse varsinaisesti asentaa mitenkään, mutta C#-ohjelmointia varten pitää ladata SQLiten kotisivuilta dynaaminen luokkakirjasto System.Data.Sqlite.dll. Valitsin .NET Framework 4.0:lle sopivan vuonna 2010 julkaistun version 1.0.84.0. Ikävä kyllä Windows-pc-koneita on olemassa 32-bittisinä sekä 64-bittisinä, joten SQLite pitää ladata erikseen näille osoitteesta http://system.data.sqlite.org/index.html/doc/trunk/www/downloads.wiki. Tarvittavien pakettien nimet ovat “Precompiled Binaries for 32-bit Windows (.NET Framework 4.0)” ja “Precompiled Binaries for 64-bit Windows (.NET Framework 4.0)”.

Latauksen jälkeen siirsin nämä paketit pääohjelman projektikansion alle samaan kansioon lähdekoodi- eli .cs-tiedostojen kanssa. Molemmissa paketeissa oli SQLite.Interop.dll -tiedosto. Tein projektikansion alle alikansiot nimeltä x86 ja x64, ja lisäsin 32-bit-paketissa olevan SQLite.Interop.dll:n kansioon x86 ja 64-bit-paketissa olevan vastaavan tiedoston kansioon x64.

Pakettien mukana tullut tiedosto System.Data.SQLite.Linq.dll on valinnainen eikä sitä ole pakko käyttää, jos koodissa ei käytetä LINQ-ominaisuuksia. Oikein käytettynä LINQ (Language Integrated Query) helpottaa tietokantahakujen tekoa ja XML-tiedostojen käsittelyä. En kuitenkaan itse tarvinnut sitä tässä työssä, koska se olisi vaatinut ensin runsaasti opettelua. Kuvassa 24 on esimerkki, kuinka LINQ toimii. Koko tietokantahaku on sijoitettu var-muuttujaan, ja myös hakutulos voidaan purkaa rivi kerrallaan foreach-luupilla var-muuttujasta.

```
public void Linq6()
{
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

    var numsPlusOne =
        from n in numbers
        select n + 1;

    Console.WriteLine("Numbers + 1:");
    foreach (var i in numsPlusOne)
    {
        Console.WriteLine(i);
    }
}
```

KUVA 24. Esimerkki LINQ:n käytöstä (37)

Tietokantatiedoston luonti

Loin Visual Studio projektiin uuden luokan SQLiteController.cs, josta oli tuleva luokka, joka toimii C#:n ja SQLiten rajapintana. Tein luokkaan kaikki SQLiteen läheisesti liittyvät ohjelmakoodit. Heti luokan alussa otetaan SQLiten DLL -paketit käyttöön rivillä using System.Data.Sqlite. Tein luokan sisälle tärkeät muuttujat SQLiteä varten. Niitä ovat SQLiteTransaction-luokan olio siirto-operaatioita varten, SQLiteConnection-luokan olio SQLite-yhteyttä varten ja SQLiteCommand-luokan olio SQLite-komennoille.

Tein funktion CreateSQLiteFile, jolla luodaan SQLite-tiedosto (liite 2). Sille annetaan parametrina string-tyyppinen tiedostonimi, esimerkksi ”Tietokanta.sqlite”. Funktiossa on myös käsky, joka ottaa SQLitelle viiteavaimet käyttöön, koska ne eivät ole vakiona päällä. Lisäksi funktio alustaa yhteyden luotuun SQLite-tiedostoon. Seuraavana tietokantaan luodaan tauluja.

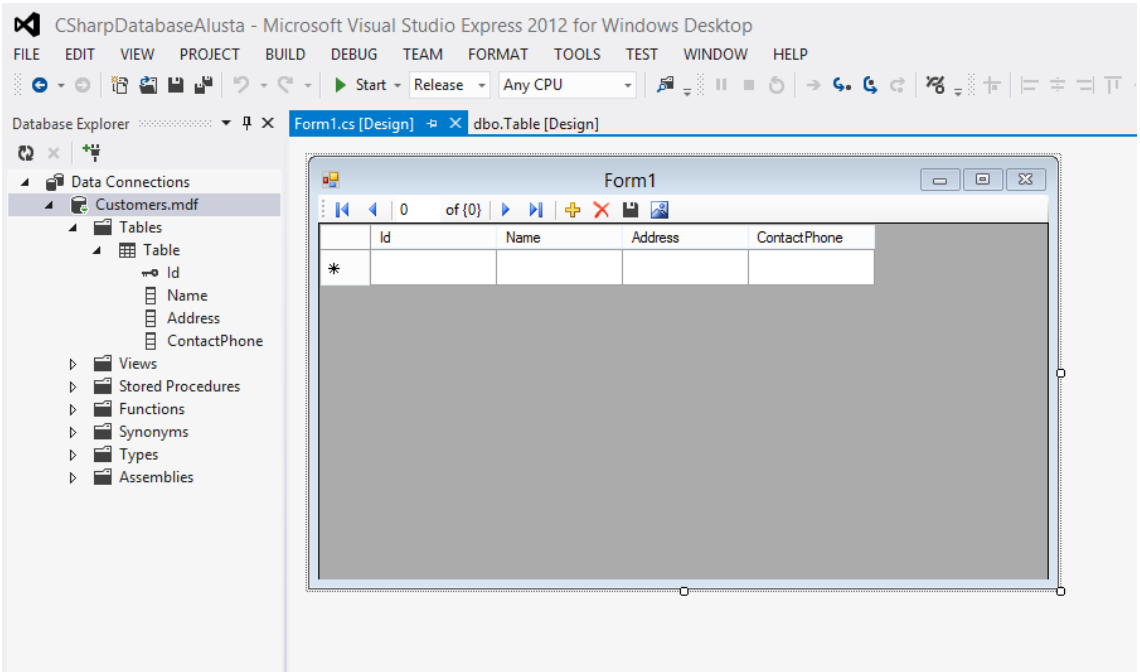
Taulun luonti tietokantaan

Heti kun sqlite-tiedosto on saatu luotua, luodaan kaikki tarvittavat taulut tietokantaan. Taulun tietokantaan luomista varten tein funktion CreateTable, jolla voidaan lisätä yksi taulu tietokantaan (liite 3). Todellisuudessa minulla oli tässä kaikkien taulujen luonti switch-case-rakenteessa, mutta tilansäästön vuoksi muutin tätä työtä varten funktion yhden taulun luonniksi.

Funktio toimii hyvänä esimerkkinä, kuinka yksi SQLite-komento toteutetaan turvallisesti try-catch-rakenteen avulla. Try-catch-rakenne tarkoittaa sitä, että try-lohkossa olevat koodit yritetään suorittaa, mutta mikäli siinä ei onnistuta, suoritetaankin catch-lohkossa olevat koodirivit. Catch-lohkolle annetaan suluissa tietyntyyppinen informatiivinen virheilmoitusparametri, joka voidaan tulostaa. Parametri voi olla esimerkiksi tyyppiä SQLiteException, joka kertoo SQLiten hakulauseen epäonnistuessa suunnilleen mistä virhe johtui.

7.3.3 C#-testisovellus Microsoft SQL Server CE -tietokannalle

Kolmanneksi kokeilin, kuinka C#:ssa tehdään tietokanta, joka on tyyppiä Microsoft SQL Server CE. Tein sitä varten erillisen C#-testisovelluksen (kuva 25). Tämän tietokannan luominen oli helppoa eikä se vaatinut erillistä palvelinta, mutta SQL-syntaksi osoittautui itselleni vaikeaksi. Syntaksi olisi vaatinut ylimääräistä opettelua. Siitä syystä hylkäsin ajatuksen tämän tietokannan käytöstä työssä heti samana päivänä, kun aloitin sen testailun.



KUVA 25. C#-testisovellus Microsoft SQL Server CE -tietokannalle

7.4 Sprint 1

Sprintin 1 alussa suunnittelin SQLite-tietokantaani tietokantasuunnittelun sääntöjen mukaan. Suunnitteluun kuului yleinen suunnittelu käsiteanalyysin

avulla, tarkempi tarveanalyysi ja tietokannan normalisointi. Tämän jälkeen aloin jalostamaan SQLiteController.cs-luokkaan tekemiäni SQLiteen liittyviä funktioita. Integroin eli liitin SQLiteController-luokkani osaksi pääsovellusta. Samalla aloittelin mittausten lisäämistä simuloimalla rivejä tietokantaani, Tein myös mittausrivien tietokantahakuja. Hakutulokset sijoitetaan C#:n taulukkoon (DataGridView-kontrolliin) ja ZedGraph-graafiin piirrettäväksi. Lisäksi lisäsin yksittäisen rivin lisäämiselle funktion sarjaporttikomponenttiin. Kokeilin työkaverin kanssa GIT-versionhallintaa, mutta siitä luovuttiin osaamattomuuden ja siitä koituneen liiallisen ajankulutuksen vuoksi (taulukko 16). Sprintille 1 annettiin taulukon 17 mukainen maali.

TAULUKKO 16. Tuotteen ominaisuuslista Scrumin Sprintissä 1

Ominaisuuden ID	**Ominaisuuden kuvaus**
7	SQLite-tietokannan suunnittelu
8	SQLite-tietokantakomponentin funktiot
9	SQLiten integrointi pääohjelmaan
10	Datojen ottaminen tiedostosta tietokantaan
11	Sarjaportti-komponentista otettavien datojen laittaminen suoraan tietokantaan
12	Graafin piirtäminen tietokannasta haetulla datalla
13	XMLParser-komponentin luonti ja integrointi pääohjelmaan Talon perustietojen säilytystä varten
14	GIT-versionhallinta
	Muut asiat

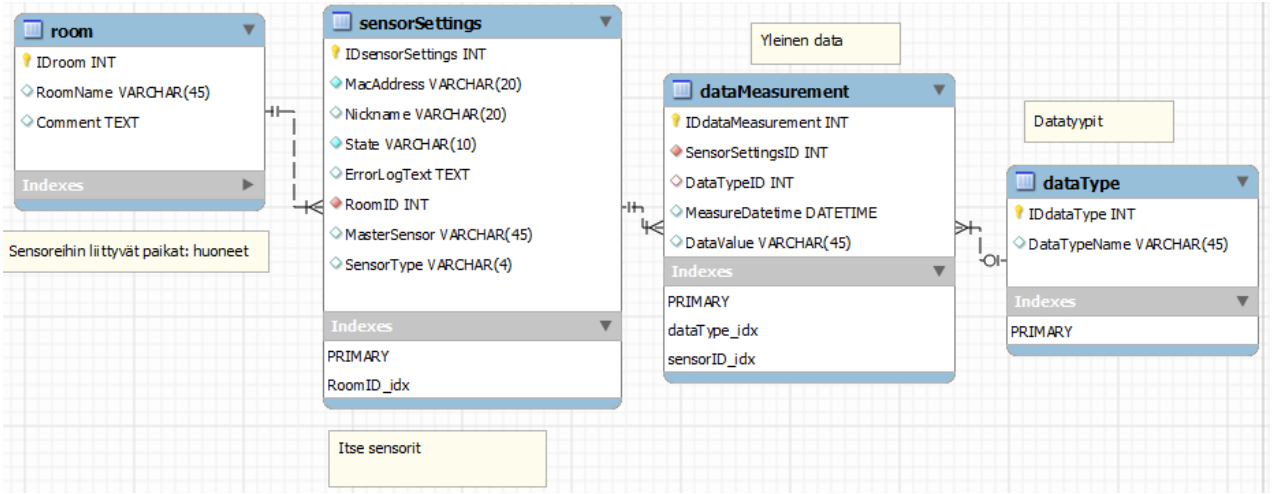
TAULUKKO 17. Sprintin 1 maali

	SPRINTIN 1 MAALI: Tietokannan vaatimuslista kirjotettuna, tietokannalla jonkinlainen mallikaavio valmis, huollon tietokanta toteutettuna, sqlite integroituna pääohjelmaan ja perusfunktiot kunnossa

7.4.1 Tietokannan suunnittelu

Suunnittelin neljä tietokantataulua ER-kaavioon. Taulut ovat room, sensorSettings, dataMeasurement ja dataType (kuva 26). Tuotteen ostaessaan käyttäjä ostaa sensoreita ja yhdistää ne ohjelmistoon langattomasti. Ohjelmisto

havaitsee uuden sensorin liittyneen verkkoon ja antaa sille automaattisesti laiteläheisen MAC-osoitteen ja sensorin tyypin. Tämän jälkeen ohjelma kysyy käyttäjältä mihin huoneeseen sensori halutaan sijoittaa ja mikä sensorin lempinimeksi annetaan. Mikäli huonetta, johon käyttäjä haluaa sensorin lisätä, ei ole, käyttäjä voi ensin lisätä haluamansa huoneen ja vasta sen jälkeen liittää uuden sensorin tähän huoneeseen.



KUVA 26. ER-kaavio, josta ilmenee suunnittelemani tietokannan rakenne tauluineen

Esitelty taulurakenne edistää hyvin aikaisemmin esiteltyä valintakriteeriä skaalautuvuus. Järjestelmään voidaan lisätä datatyyppejä, sensoreita, huoneita ja mittaustuloksia ilman että tietokantahaut menisivät toimimattomaksi. Esimerkiksi voidaan lisätä rajattomasti datatyyppejä tai huoneita ennenkuin yhtään sensoria tai sensorin mittaustulosta lisätään järjestelmään.

Taulujen rivimäärien suhteet ovat erilaiset. Tauluista selvästi eniten tietoa sisältävä on dataMeasurement-taulu, johon voi tulla satoja tuhansia rivejä. Koska taulu on niin suuri, on järjestelmää varten luotava esimerkiksi puolivuosittain uusi SQLite-tietokantatiedosto. Room-tauluun tulee rivejä yhtä monta kuin järjestelmään on tallennettu huoneita. DataType-tauluun tulee rivejä sen mukaan, kuinka paljon erilaisia mittaustuloksia järjestelmään lisätyt sensorit kykenevät lähettämään. Luultavasti parhaassakin tapauksessa alle dataType-taulun rivit jäävät alle kymmeneen. SensorSettings-tauluun tulee rivejä yhtä monta kuin sensoreita on lisätty järjestelmään.

SensorSettings-taulu

SensorSettings-taulu sisältää tiedot sensoreista. Tiedot ovat sensorin ID, mac-osoite eli laitteistokohtainen yksilöllinen pitkä ID-lukema jolla sensorit tunnistetaan oikeaksi, lempinimi, sensorin tila, viimeisin virheviesti, huoneen ID eli huone mihin sensori on asetettu, mastersensor-merkkijono, joka kertoo, onko kyseessä sensori, jonka tietoja näytetään pääohjelman etusivulla, ja sensorin tyyppi, koska järjestelmään voi ostaa erityyppisiä sensoreita. Huoneen ID kenttä ei saa olla NULL, koska jokainen sensori on pakko olla jossain huoneessa. Sensori, joka mittaa ulkotiloja täytyy lisätä myös huoneeseen, joka on esimerkiksi ”ulkotila”. Mastersensoreita on yksi jokaista sensorin tyyppiä kohti. Sensorin tyyppiä ei tule sekoittaa datatyyppiin, vaan esimerkiksi sensorityyppiä A1 sensori voisi mitata ainoastaan lämpötilaa ja A2 lämpötilaa ja jotain toista arvoa. Sensorin tyyppi on siis vain lisäominaisuus hakuja varten.

Room-taulu

Room-taulussa on tallessa huoneiden tiedot. Tietoja ovat huoneen ID, huoneen nimi ja mahdollinen kommenttiteksti. Huonetietoja käytetään tietokantahakuihin. Käyttäjä voi valita näytettäväksi vain tietyn huoneen sensorit käyttöliittymän graafilla tai taulukossa.

DataType-taulu

DataType-taulussa on tallessa erilaiset mittaustyypit. Taulussa on valmiina tuotetta ostaessa erilaisia datatyyppejä, esimerkksi lämpötila. Tyypeillä on ID ja nimi. Tyyppejä voidaan tulevaisuudessa helposti lisätä ja poistaa, eikä mittaustulokset-taulu häiriinny.

DataMeasurement-taulu

DataMeasurement-taulu sisältää kaikki mittaustulokset. Taulussa on viittaus tiettyyn sensoriin sensorin ID:llä ja viittaus tiettyyn datatyyppiin, esimerkiksi lämpötila, datatyypin ID:llä. Lisäksi taulussa on mittaustuloksen arvo liukulukuna, esimerkiksi 23.0, ja päivämäärä-kellonaika-yhdistelmä, josta ilmenee mihin aikaan mittaus saatiin, esimerkiksi 16.3.2013 11:38:00.

Päivämäärän formaattina käytin SQLitessä ”dd-mm-YYYY HH:MM:ss”. Esimerkkikellonaika on tällä formaatilla ilmaistuna 16-03-2013 11:38:00.

7.4.2 Säie suorittamaan hakulauseita

Hakutapahtumat ovat SQL-kielellä tehtäviä SELECT-lauseita, jotka noutavat tietokannasta halutun tiedon. Koska hakutapahtumat kestävät suurella määrällä dataa monta sekuntia, tai jopa kymmeniä sekunteja, loin SQLite-tapahtumia varten uuden oman C#-luokan SQLiteWorkerThread, jossa on säie (engl. thread). Säikeen avulla hakulause voi tapahtua taustalla ilman, että ohjelman pääsäie hidastuu paljoa eli ohjelman käyttö, johon kuuluu esimerkiksi nappien painallukset ja näkymien vaihdot, ei hidastu. Tietokoneen prosessori antaa vuorotellen ajoaikaa pääsäikeelle luomalleni apusäikeelle.

Tein säikeeseen muteksin (engl. mutex), jonka ansiosta säieluokkaa SQLiteWorkerThread ei voi käyttää yhtä aikaa kuin yksi säiefunktio, joten SQLite-tiedoston lukitus ei ole enää ongelma.

Määrittelin SQLiteWorkerThread-luokan sisällä tarvittavat oliot. Olioita olivat Thread-luokan olio säikeelle, Mutex-luokan olio muteksille, omatekoisen SQLiteController-luokkani olio SQLite-komennoille ja Main-luokan olio, joka sisältää kopion pääluokasta. Kuvassa 27 on SQLiteWorkerThread-luokan muodostin.

Main-oliota tarvitaan, koska SQLiteControllerille pitää lähettää muodostinfunktiossa Mainin kopio, koska SQLiteControllerista kutsutaan suoraan Mainissa sijaitsevia delegaattifunktioita. Delegaattia tarvitaan sen erikoisen luonteen takia.

```
//SQLiteWorkerThread-luokan muodostin ottaa parametriksi Main-luokan ja
sieltä saapuvan muteksin
public SQLiteWorkerThread(Main thisMain, Mutex _globalMutex)
{
  //otetaan kopiot Main-oliosta ja Mutex-oliosta paikallisiin olioihin
  oMain = thisMain;
  myCopyOfMutex = _globalMutex;
  //luodaan sqlite-olio SQLiteController-luokasta parametrinaan Main-olion
  paikallinen kopio oMain.
  sqlite = new SQLiteController(oMain);
}
```

KUVA 27. SQLiteWorkerThread-luokan muodostin

Tein SQLiteWorkerThread-luokkaan säikeenkäynnistysfunktion StartTheThread, joka ottaa parametriksi erilaisia tyyppejä, joista tässä esimerkin vuoksi vain lämpötilan lisääminen parametrillä ”InsertTemperature” (kuva 28). Funktio on tyyppiä public eli julkinen, joten sitä voidaan kutsua esimerkiksi Main-luokasta siellä luodulla SQLiteWorkerThread-oliolla.

```
//Säikeenkäynnistys-funktio, jota kutsutaan muista luokista tietyllä parametrilla
public void StartTheThread(string workerThreadFunctionType)
{
  //Tutki parametriä. Jos se on esimerkiksi InsertTemperature, käytä
  //WorkerThreadFunctionInserttiä ja anna sille parametriksi Startissa
  //temperature.
  if (workerThreadFunctionType == "InsertTemperature")
  {
    //Luo parametrillinen säie, jonka suluissa annetaan säiefunktio
    workerThread = new Thread(new
      ParameterizedThreadStart(this.WorkerThreadFunctionInsert));
    //Säikeelle yksilöllinen nimi debuggausta varten
    workerThread.Name = "Database Worker Thread Insert Temperature";
    //säikeelle parametri stringinä
    workerThread.Start("temperature");
  }
}
```

KUVA 28. StartTheThread-funktio, jossa käynnistetään parametrillinen säie

7.4.3 Mittauksien lisääminen tietokantaan

Kun tietokanta ja sen taulut luodaan sarakkeineen, tauluissa ei ole aluksi rivejä. Rivejä voidaan lisätä tietokantatauluun antamalla jokaiselle riville arvoja yhtä monta kuin taulussa on sarakkeita. Arvon täytyy olla sarakkeen tyypin mukainen ja sen maksimipituuden rajoissa.

Mittaustuloksia lisätään dataMeasurement-tauluun SQLiteWorkerThread-luokan avulla säikeen käynnistyksen jälkeen funktiossa WorkerThreadFunctionInsert. Tein testaamista varten tiedonsimulointimahdollisuuden, joka simuloi annetun määrän rivejä kerrallaan transaktiolla eli siirto-operaatiolla. Simuloitu data on keksittyä tai arvottua, ei-oikeata tietoa. WorkerThreadFunctionInsert-funktio lisää annetun määrän verran rivejä tietokantaan tietylle datatyypille. Tosin tässä esimerkin vuoksi käytin taas vain lämpötilan lisäystä (liite 4).

7.4.4 Taulun tyhjentäminen tietueista

Kun tauluihin lisättiin suuri määrä tietuita, piti jo aikaisessa vaiheessa miettiä toteutusta, jolla voitaisiin tyhjentää tauluja tietueista. Kokonaisen taulun tyhjentämiseen tarvitaan SQLitessä kolme käskyä. Otetaan esimerkkinä dataType-taulun tyhjentäminen. Ensimmäinen käsky tarkoittaa, että poistetaan kaikki rivit taulusta nimeltä dataType (kuva 29).

```
"DELETE FROM 'dataType';"
```

KUVA 29. DataType-taulun tyhjentävä SQL-lause

Toinen käsky tyhjentää sqlite_sequence -taulusta kaikki viittaukset dataType-taulusta. Lisäksi se resetoi auto_increment-tyyppisen ID-kentän asettamalla sen NULL-arvoon. (Kuva 30.)

```
"DELETE FROM sqlite_sequence WHERE name='dataType';"
```

KUVA 30. SQLiten SQLite_sequence taulun tyhjentäminen dataType-viittauksista

Kolmas tarvittava käsky on: `”VACUUM;"` Aiemmat kaksi komentoa jättivät NULL-arvoja tietokantaan. Vacuum ajaa SQLite-tiedostolle tarpeellisen NULL-arvojen

poiston, jolloin sen fyysinen tiedostokoko pienenee, koska NULL-arvotkin vievät aina tilaa. Eron huomaa hyvin, kun ensin tallennetaan testin vuoksi miljoona riviä tietokantaan, poistetaan nämä rivit ja ajetaan vacuum-komennon. Ennen vacuumia tiedoston koko säilyi suurena, ja sen jälkeen koko oli pienentynyt.

7.5 Sprint 2

Sprintissä 2 jatkoin graafin piirtämistä tietokannasta haetulla datalla, parantelin datan hakulauseita entisestään ja aloin keskittyä yhä enemmän UI-toteutukseen. Ominaisuudet on listattu taulukkoon 17 ja sprintille annettiin taulukon 18 mukainen maali.

TAULUKKO 17. Tuotteen ominaisuuslista Scrumin Sprintissä 2

Ominaisuuden ID	Ominaisuuden kuvaus
10	Datojen ottaminen tiedostosta tietokantaan
12	Graafin piirtäminen tietokannasta haetulla datalla
15	Datan hakeminen ja käyttäminen
16	UI toteutus
	Muut asiat

TAULUKKO 18. Sprintin 2 maali

	SPRINTIN 2 MAALI: Sensoridataa alustavasti tietokannassa, sensoridatan haku aikaleimalla pääohjelmalle toiminnassa, tietokannan testaus

7.5.1 Monen rivin hakeminen graafia varten

Toteutin hakulauseen toteuttamallani funktiolla ThreadSelectDataMeasurements (liite 5), jolla haetaan massiivisesta määrästä rivejä vain tietty määrä graafia varten. Ohjelmassa käyttäjä voi valita ensinnäkin tilan jossa sensori sijaitsee, mittausajankohdan (esimerkiksi 12.3.2013, jos käyttäjä haluaa tarkastella sen päivän mittauksia), joka määrittää mikä on hakulauseeseen sisällytetty viimeinen päivämäärä, näytettävän aikavälin (tunti, viikko, kuukausi jne.) sekä parametrin jota haetaan (esimerkiksi lämpötila).

Haku suoritetaan transaktiolla säikeen avulla ja tiedot palautetaan DataSet-tyyppiseen olioon, josta ne pilkotaan tavallisiksi taulukoiksi ja piirretään graafille.

7.5.2 DataSet-muuttuja hakutulosten säilytykseen

Tekemälleni funktiolle GetDataSetWithQuery annetaan tietokantakysely merkkijonona eli string-parametrina. Kyselystä tieto saadaan SQLiteDataAdapter-luokan avulla talteen DataSet-tyyppiseen muuttujaan. Näin haettua tietopakettia voidaan nyt liikutella koodissa paikasta toiseen vain yhtä ainoaa DataSet-muuttujaa välittämällä. GetDataSetWithQuery sisältää myös C#:n StopWatch-luokasta luodun ajastimen nimeltä timer. Timer mittaa hakuun kulunutta aikaa ja ilmoittaa sen debuggaus-ikkunassa kun haku on päättynyt. (Kuva 31.)

```
//Funktio joka hakee annetulla hakulauseella tietokannasta rivejä ja palauttaa ne
//DataSet-muuttujana kutsukohtaan
public DataSet GetDataSetWithQuery(string query)
{
  //Timerin starttaus
  timer.Reset();
  timer.Start();
  Debug.WriteLine("Timer alkaa mittaamaan: ");
  SetConnection();

  DataSet dataSet = new DataSet();

  try
  {
    var dataAdapter = new SQLiteDataAdapter(query, conn);
    dataAdapter.Fill(dataSet);
  }
  catch (SQLiteException ex)
  {
    MessageBox.Show("Get data function fails " + ex.ToString());
  }
  finally
  {
    if (conn != null)
```

```
        conn.Close();
    }

    timer.Stop();
    Debug.WriteLine(timer.ElapsedMilliseconds.ToString("Time taken for DataSet
                return " + "#,##0.00 'milliseconds'"));
    return dataSet;
}
```

KUVA 31. GetDataSetWithQuery-funktio

Funktio IsEmpty tutkii onko sille annettu DataSet tyhjä vai ei (kuva 32). Jos annettu DataSet on tyhjä, ei hakulause tuottanut yhtään hakutulosta tai on virheellinen. Käytän tätä funktiota useassa paikassa koodissa. Ensin haetaan vaikka huoneen nimellä huoneen ID. Toinen haku tehdään heti tämän jälkeen: etsitään sensoria, jolla on kyseisen huoneen ID. Toista hakua ei kannata yrittääkään tehdä, jos ensimmäinen palauttaa tyhjän tuloksen, koska tällöin toinenkin haku palauttaisi vain tyhjää ja tietokantaan tehtäisiin turhia yhteydenottoja.

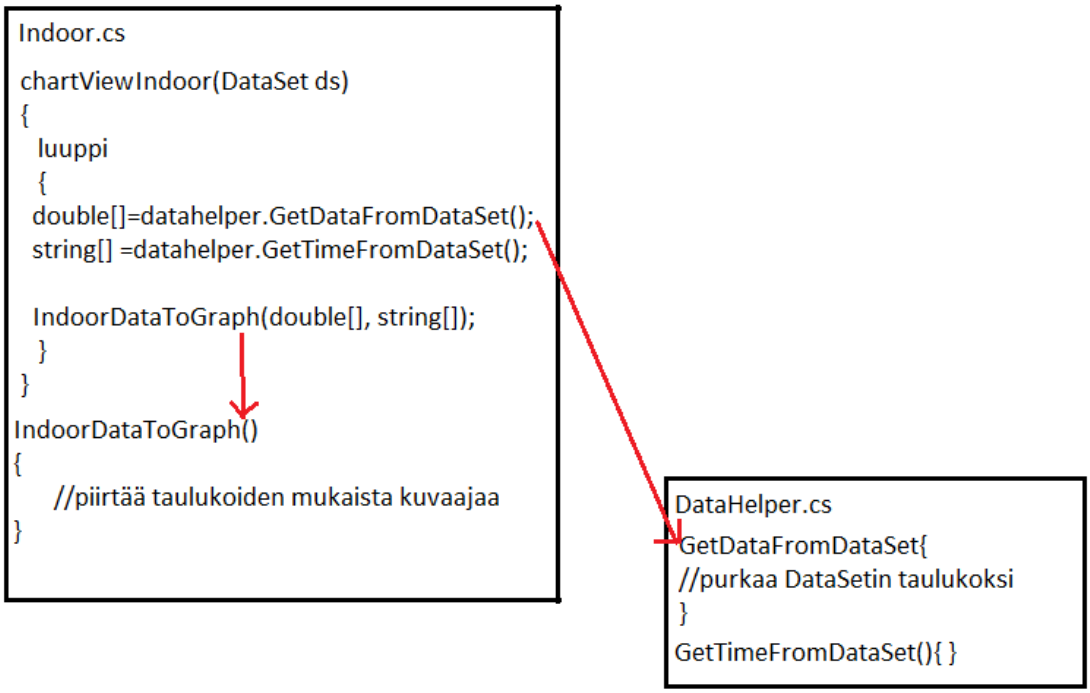
```
//funktio jolla voidaan tarkastaa onko annettu dataset tyhjä vai ei
public bool IsEmpty(DataSet dataSet)
{
    foreach (DataTable table in dataSet.Tables)
        if (table.Rows.Count != 0) return false;
    return true;
}
```

KUVA 32. IsEmpty-funktio

7.5.3 DataSetin purkaminen tavallisiin taulukoihin

Kunhan säie on hoitanut mittaustietojen hakemisen tietokannasta, se viestittää delegaatilla hakutulokset eteenpäin DataSetissä. Tällöin kutsutaan funktiota

chartViewIndoor (liite 6), joka puolestaan kutsuu funktioita GetTimeFromDataSet (liite 6) ja GetDataFromDataSet, jotka purkavat DataSetin tavallisiksi string- ja double-tauluiksi, jonka jälkeen lähettävät taulut takaisin chartViewIndoor-funktiolle. Taulut lähetetään puolestaan eteenpäin luupin sisältä graafifunktiolle nimeltään IndoorDataToGraph. Rakennetta yritetään selventää kuvassa 33 pseudokoodilla.



KUVA 33. Pseudokoodia, siitä miten DataSet puretaan taulukoiksi ja niistä piirretään puolestaan graafia

7.5.4 ZedGraph-graafin piirtäminen

Viimeinen vaihe tietojen saattamisessa graafille on siis chartViewIndoor-funktion sisältä kutsuttu funktio IndoorDataToGraph. Tämän funktion parametrit ovat aikaleimoja sisältävä string-taulukko timeArray, mittausarvoja sisältävä double-taulukko dataArray ja datan tyyppi eli string dataType. Funktiossa asetetaan paljon ZedGraphin asetuksia. Paneelia ja sen akseleita skaalataan, fonttikokoja muutetaan ja graafiin piirretään lopulta taulukkojen mukainen kuvaaja. (Liite 7.)

7.6 Sprint 3

Sprint 3:ssa korjailin graafia ja pääohjelman käyttöliittymää ja tein paljon koodin siivousta ja uudelleen toteutusta (taulukko 19). Jonkin verran aikaa meni siihen, että ohjelmisto pyrittiin julkaisemaan asennus- eli deployment-pakettina kaikkine tarpeellisine DLL-kirjastoineen Visual Studiosta. Sprintille 3 annettiin taulukon 20 mukainen maali.

TAULUKKO 19. Tuotteen ominaisuuslista Scrumin Sprintissä 3

Ominaisuuden ID	Ominaisuuden kuvaus
17	*Graafin fixaus*
18	*UI ja siihen liittyvät asiat*
19	*Visual studiosta deployment-paketin tekeminen sekä 32bit että 64bit järjestelmiin*
	Muut asiat

TAULUKKO 20. Sprintin 3 maali

	SPRINTIN 3 MAALI: Valmis hyvin pääohjelman kanssa toimiva tietokanta, toimivat käyrien piirrot graafeissa

7.6.1 Uusimman mittaustuloksen hakeminen

Koodasin hakulauseen muun muassa uusimman mittaustuloksen hakemiseksi tietylle sensorille tietylle datatyypille tietyssä huoneessa. Tämä toteutuu funktiossa SelectLatestValue (kuva 34). Kyseessä on neljän taulun yhdistelmä JOIN-lauseella, ja lisäksi SQL-lause sisältää alikyselyn, joka etsii max-funktiolla suurimman eli uusimman arvon mittausajalle.

Koska alikysely saattaa palauttaa useamman kuin yhden rivin siitä syystä, että mittaustuloksia saattaa tulla samalla aikaleimalla useampia tietokantaan, pitää vielä kirjoittaa hakuehtoja WHERE-sanan jälkeen. Ehtona on, että huoneen ID on parametrina annettava string roomID ja sensorin lempinimi (Nickname) on parametri string nickname sekä datatyypin nimi (DataTypeName) on parametri string dataTypeName.

```
//uusimman mittaustuloksen hakeminen
public void SelectLatestValue(object dataTypeName, object roomID, object
nickname)
{
//Muteksin lukitseminen
myCopyOfMutex.WaitOne();
//Toimiva "viimeisimmän arvon haku" -query
string query = "SELECT dt.MeasureDatetime, dt.DataValue " +
"FROM dataMeasurement AS dt, (SELECT max(MeasureDatetime) as
maxdate, SensorSettingsID, DataTypeID FROM dataMeasurement GROUP BY
SensorSettingsID, DataTypeID) AS maxresults " +
"JOIN dataType AS tyypit ON dt.DataTypeID = tyypit.IDdataType "+
"JOIN sensorSettings AS sensorit ON dt.SensorSettingsID = senso-
rit.IDsensorSettings " +
"JOIN room AS huoneet ON sensorit.RoomID = huoneet.IDroom " +
"WHERE dt.DataTypeID = maxresults.DataTypeID AND dt.MeasureDatetime =
maxresults.maxdate AND dt.SensorSettingsID = maxresults.SensorSettingsID "
+
"AND huoneet.IDroom = '"+ (string)roomID + "' " +
"AND sensorit.Nickname = '" + (string)nickname + "' " +
"AND tyypit.DataTypeName = '" + (string)dataTypeName + "' " +
"ORDER BY dt.IDdataMeasurement ASC;";

//SQLiteController-luokassa haetaan SQLite-tietokannasta annetulla
hakulauseella DataSet ja palautetaan se tähän.
DataSet ds = sqlite.GetDataSetWithQuery(query);

//Jos publicDemo-olio on olemassa ja sillä on tarvittava kahva, lähetetään
DataSet nimeltä ds sekä string-arvot dataTypeName ja roomID eteenpäin
delegaatin avulla publicDemo-luokkaan
if(!oPublicDemo.IsDisposed && oPublicDemo.IsHandleCreated)
  oPublicDemo.Invoke(oPublicDemo.publicDelegateLatestValues,
  ds(string)dataTypeName, (string)roomID);

//Muteksin vapauttaminen
myCopyOfMutex.ReleaseMutex();
}
```

KUVA 34. SelectLatestValue-funktio

7.6.2 Tietokannan optimointi

Tietokannan optimointi on suurille tietomäärille välttämätön toimenpide. Optimointi tarkoittaa tehokkuuden parantamista. SQLite-tietokantaa voidaan optimoida tietyin toimenpitein. Mittasin tiettyihin tietokantahakuihin ja -lisäyksiin käytettyä aikaa jo mainitsemallani C#:n Stopwatch-luokasta luodulla oliolla. Tein työssä optimointia luomalla kentille indeksejä ja säätämällä SQLiten käynnistysasetuksia, optimoin hakulauseita ja suunnittelin tietokannan osittamista. Osittaminen jäi työssä kuitenkin toteuttamatta.

Indeksointi

Indeksointi tarkoittaa SQL-tietokannassa ikään kuin sitä, että kirjassa on sisällysluettelo. Indeksin avulla haettu data löytyy siis nopeampaa kuin ilman indeksiä. Ei tarvitse selata koko kirjaa läpi rivi riviltä, kun jo sisällysluettelosta nähdään, missä mikäkin sijaitsee.

Indeksit luodaan automaattisesti ID- ja viiteavain-kentille tietokannassa. Päätin kuitenkin luoda indeksin lisäksi dataMeasurement-taulun jäsenille, joita käytetään usein hakulauseissa. Näitä olivat measureDatetime ja dataValue. Esimerkkinä tässä SQL-komento joka luo indeksin dataValue-kentälle: ”CREATE INDEX IF NOT EXISTS dataValueIndex ON dataMeasurement (dataValue ASC)”.

SQLiten asetukset

Käytin SQLiten luontivaiheessa connection-stringiä eli merkkijonoa, joka määrittää SQLitelle tiettyjä perusasetuksia. Näitä olivat mm. seuraavat:

1. PRAGMA synchoronous=off;
2. PRAGMA cache_size= 20000;
3. PRAGMA page_size=32768;
4. PRAGMA journal_mode=off;

Asetusten ideana on poistaa synkkaus ja journal mode, joka hidastavat SQLite-operaatioita ja kasvattaa cachen eli välimuistin ja pagejen eli sivujen kokoa.

Asetuksien vaikutusta testattiin työssä Stopwatch-ajastimen avulla, mutta ne eivät näyttäneet nopeuttavan hakuja.

Hakulauseiden optimoinnista

Hakulauseissa kannattaa valita vain sarakkeet mitä oikeasti tarvitaan ja valittavat sarakkeet on laitettava samaan järjestykseen kuin ne ovat taulussa.

Tietokannan osittaminen

Koska halutaan pääohjelman näyttävän sekä reaaliaikaista dataa sekä aikaisemmin mitattuja historiadatoja valitulta aikaväliltä, päätettiin luoda useita tietokantoja, mikä tarkoittaa käytännössä useita SQLite-tiedostoja.

Reaaliaikatietokantoja luodaan joka sensorille oma tiedosto, jossa säilytetään 5 sekunnin välein mitattua dataa ainoastaan yhden tunnin ajalta. Tämä on ikäänkuin puskuritietokanta, josta tyhjennetään yli tunnin vanhat datat aina pois. Käyttöliittymässä reaaliaikainen tieto tarkoittaa, että joka 5 sekunnin välein päivitetään graafiin uusi piste viivan jatkoksi ja maksimissaa graafilla näytetään mittaustuloksia tunnin ajalta. Dataa tulee näin ollen tietokantaan ja graafille korkeintaan 720 kpl, koska minuutissa tulee 60 / 5 = 12 kpl ja tunnissa 60 x 12 = 720 kpl. Näin vähän rivimäärän haku on hyvin nopea eikä vie paljoa tilaa. Jos sensorille ei löydy dataa viimeisen tunnin ajalta, aloitetaan piirtäminen tyhjään graafiin.

Historiatiedot haetaan isommasta tietokannasta jonne tallennetaan vain noin 15 minuutin välein ohjelmalle sensorilta saapuva mittaustulos. Tämä siksi että pitkän aikavälin historiadata ei mahtuisi muutenkaan kovin yksityiskohtaisesti graafille. Esimerkiksi kuluneen vuoden datat näyttävässä graafissa riittää kun näytetään joka kuukaudelle keskiarvoinen tulos tai pari.

Kun ei talleteta edes joka minuutilta dataa (mikä oli alunperin suunnitelmana) vaan vain 15 minuutin välein, säästyy tilaa 15 kertainen määrä ja hakulauseet nopeutuvat 15-kertaisesti. Tällaiseen tietokantaan kertyy rivejä vuodessa yhdeltä sensorilta 4 * 24 * 365 = 35040 kappaletta mikä tarkoittaa alle sekunnin hakuoperaatiota, jolloin käyttäjä pysyy tyytyväisenä.

8 YHTEENVETO

Tämä työ oli varsin haastava. Tiesin vain vähän MySQL-tietokannasta aiempien kokemusten perusteella. Aloitin työn tietokannan suunnittelusta. Myöhemmin tietokantakatselmointipalaverin jälkeen jouduin kuitenkin muuttamaan tietokannan rakennetta. Sitten aloin toteuttaa SQLite-tietokannan ja C#:n rajapintaa ja SQL-hakulauseita. Myöhemmin pääpaino siirtyi pääohjelman tekemisessä UI:n tekoon.

Osasin kohtuullisen hyvin C#-koodia ennestään, mutta tässäkin työssä alkoi korostua se miten vaikeaa on suunnitella arkkitehtuurisesti hyvää koodia. Lisäksi olio-ohjelmointi tuli entistä tutummaksi. En ole vieläkään täysin tyytyväinen tekemiini koodiratkaisuihin, mutta toivottavasti opin pian korjaamaan ne. Olisin voinut myös kommentoida koodia paremmin.

Mielestäni Scrum ei kovin hyvin soveltunut tähän opinnäytetyöhön, koska se toimii parhaiten, jos on enemmän kuin yksi tekijä, jolloin töitä ja osaamista voidaan jakaa. En myöskään oikein osannut suunnitella taskeihin käytettäviä tuntimääriä, koska en osannut tekemiäni asioita juurikaan ennen työtä. Scrumin päiväpalaverit pidettiin minun, Scrum-mestarin ja toisen koodarin välillä päivittäin, ja niissä sitten integroimme tekemiämme koodipätkiä yhteen, mutta esimerkiksi Scrum-katselmoinnit jäivät aika vähälle jatkuvan kiireen takia.

Yritimme käyttää alussa GIT-versionhallintatyökalua, mutta pian osaamattomuutemme alkoi tulla vastaan, kun GIT ilmoitti yhdistämisen konflikteista ja niiden purkamiseen meni liikaa aikaa. Siirryimme käyttämään WinMergeä koodien yhdistelyyn. Otimme tiedostoista varmuuskopioita ennen WinMergellä yhdistelyä.

Mielestäni onnistuin työssäni ihan hyvin. Lopulta tietoa saatiin lisättyä tietokantaan ja haettua oikealla lailla tietokannasta graafille ja tekstikenttiin. Minua jäi vain vaivaamaan tietokannan kesken jäänyt tehokkuuden optimointi. Sensorit saattavat lähettää vuodessa monta miljoonaa mittaustulosta. Simuloin tällaisia rivimääriä tietokantaan. Totesin suuren määrän mittausrivejä hidastavan tietokantaa sangen paljon.

Nykytilanteessa, kun tietokanta sisältää 500 000 riviä, kestää tiedon hakeminen graafille noin 10 sekuntia. Käyttäjä joutuu siis odottamaan 10 sekuntia, ennen kuin näkee hakutulokset, mikä on aivan liian pitkä aika nykysovelluksille. Siispä tieto pitäisi hakea jo ennen napin painallusta taustalla käyttäjän tietämättä ja näyttää silloin kun nappia painetaan. Toinen mahdollisuus on jatkossa osittaa tietokanta usean SQLite-tiedoston alle ja käyttää SQLiten ATTACH-komentoa, jolla voidaan liittää useita SQLite-tiedostoja samaan hakulauseeseen. Reaaliaikaisen tiedon näyttäminen jäi työssä vielä toteuttamatta.

Työssä koin hankalaksi hyvän koodiarkkitehtuurin toteuttamisen. Koodi ei ollut lopulta kovin muutosjoustavaa eikä se sisältänyt mielestäni tarpeeksi oliomalliin pohjautuvaa ajattelua. Koodia voisi kutsua paikoitellen spagettikoodiksi, koska siinä on vaikea pysyä kärryillä, että mihin mikäkin koodin pätkä johtaa. Koen kuitenkin, että kehityin tällä alueella kokoajan työn edetessä.

Lisäksi mainittakoon, että Visual Studio 2012:sta oli harvinaisen vaikeaa tehdä sekä 32- että 64-bittiset käännökset siten, että SQLite-luokkakirjaston sai mukaan helposti. Ainakin Visual Studion Express-versiosta puuttui valmiin asentajan (engl. installer) luominen. Vasta työn päätyttyä myöhemmin löysin tämän linkin http://www.tsjensen.com/blog/post/2012/11/10/SQLite-on-Visual-Studio-with-NuGet-and-Easy-Instructions.aspx. Linkissä kerrotaan miten helpoksi paketin luominen on tehty Visual Studioon asennettavalla lisäohjelmalla nimeltä NuGet.

LÄHTEET

1. Anturi. 2013. Saatavissa: http://fi.wikipedia.org/wiki/Anturi. Hakupäivä 21.3.2013.

2. Visual Studio. 2013. Saatavissa: http://fi.wikipedia.org/wiki/Visual_Studio. Hakupäivä 10.3.2013.

3. .NET Framework. 2013. Saatavissa: http://fi.wikipedia.org/wiki/.NET_Framework. Hakupäivä 21.3.2013.

4. Onko tietokoneessani käytössä 32-bittinen vai 64-bittinen Windows-versio?. 2013. Saatavissa: http://windows.microsoft.com/fi-fi/windows7/find-out-32-or-64-bit. Hakupäivä 21.3.2013.

5. C sharp. 2012. Saatavissa: http://fi.wikipedia.org/wiki/C_sharp. Hakupäivä 28.2.2013.

6. Lähdekoodi. 2013. Saatavissa: http://fi.wikipedia.org/wiki/L%C3%A4hdekoodi. Hakupäivä 15.3.2013.

7. Debuggaus. 2012. Saatavissa: https://trac.cc.jyu.fi/projects/ohj1/wiki/debuggaus. Hakupäivä 21.3.2013.

8. Diagrammi. 2013. Saatavissa: http://fi.wikipedia.org/wiki/Diagrammi. Hakupäivä 9.4.2013.

9. ZedGraph. 2013. Saatavissa: http://zedgraph.sourceforge.net/index.html. Hakupäivä 21.3.2013.

10. Kuva: ZedGraph-esimerkki. 2007. Saatavissa: http://www.voidspace.org.uk/python/weblog/arch_d7_2007_06_16.shtml. Hakupäivä 17.4.2013

11. About WinMerge. 2013. Saatavissa: http://winmerge.org/about/. Hakupäivä 28.2.2013.

12. WinMerge-manual: Merging files. 2013. http://manual.winmerge.org/Intro_diffs.html#Intro_diffs_merging. Hakupäivä 21.3.2013.

13. Ohjelmiston versionhallinta. 2012. Saatavissa: http://fi.wikipedia.org/wiki/Ohjelmiston_versiohallinta. Hakupäivä 28.2.2013.

14. SQLite Database Browser. 2009. Saatavissa: http://sqlitebrowser.sourceforge.net/. Hakupäivä 21.3.2013.

15. MySQL Workbench. 2013. Saatavissa: http://www.mysql.com/products/workbench/. Hakupäivä 8.3.2013.

16. Projektinhallinta. 2013. Saatavissa: http://fi.wikipedia.org/wiki/Projektinhallinta. Hakupäivä 21.3.2013.

17. Scrum. 2013. Saatavissa: http://fi.wikipedia.org/wiki/Scrum. Hakupäivä 21.3.2013.

18. Kuva: Scum-prosessi. 2011. Saatavissa: http://hlab.ee.tut.fi/hmopetus/vpsist-oppimateriaali/4-menetelmia-ja-malleja/4-4-ketterat-menetelmat/4-4-1-scrum. Hakupäivä 20.3.2013.

19. Scrumin määritelmät ja pelisäännöt. 2011. Saatavissa: http://www.scrum.org/Portals/0/Documents/Scrum%20Guides/Scrum%20Guide%20-%20FI.pdf. Hakupäivä 8.3.2013.

20. Tietokanta. 2013. Saatavissa: http://fi.wikipedia.org/wiki/Tietokanta. Hakupäivä 8.3.2013.

21. Tietotyypit. 2004. Saatavissa: http://appro.mit.jyu.fi/doc/tiedonhallinta/sql/ddl/index1.html. Hakupäivä 21.3.2013.

22. SQL Primary Key. 2013. Saatavissa: http://www.w3schools.com/sql/sql_primarykey.asp. Hakupäivä 21.3.2013.

23. Yleisimmät tietokantajärjestelmät. 2013. Saatavissa: http://dbconvert.com/overview.php. Hakupäivä 8.3.2013.

24. Sarja, Jari. 2006. Relaatiotietokanta. Saatavissa: http://www.verkkopedagogi.net/vanhat/fi/sisalto/materiaalit/access2003/luku0375c6.html?C:D=419702&selres=419702. Hakupäivä 21.3.2013.

25. Foreign Key. Saatavissa: http://www.w3schools.com/sql/sql_foreignkey.asp. Hakupäivä 21.3.2013.

26. SQL. 2013. Saatavissa: http://fi.wikipedia.org/wiki/SQL. Hakupäivä 21.3.2013.

27. SQL Functions. Saatavissa: http://www.w3schools.com/sql/sql_functions.asp. Hakupäivä 21.3.2013.

28. Tietokantasuunnittelu. 2008. Saatavissa: http://www.cs.helsinki.fi/u/ronkaine/tikape/K2008/pdf/tietokantasuunnittelu.k08.pdf. Hakupäivä 20.3.2013.

29. Hovi, Ari – Huotari, Jouni – Lahdenmäki Tapio 2005. Tietokantojen suunnittelu ja indeksointi. Jyväskylä: Docendo.

30. Entity-relationship model. 2013. Saatavissa: http://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model. Hakupäivä 21.3.2013.

31. About SQLite. 2013. Saatavissa: http://www.sqlite.org/about.html. Hakupäivä 8.3.2013.

32. Categorical Index Of SQLite Documents. 2013. Saatavissa: http://www.sqlite.org/docs.html. Hakupäivä 8.3.2013.

33. Video: An Introduction To SQLite. 2007. Saatavissa: http://www.youtube.com/watch?v=f428dSRkTs4. Hakupäivä 8.3.2013.

34. MySQL. 2013. Saatavissa: http://fi.wikipedia.org/wiki/MySQL. Hakupäivä 21.3.2013.

35. XAMPP. 2013. Saatavissa: http://www.apachefriends.org/en/xampp.html. Hakupäivä 24.3.2013.

36. Overview of SQL Server Compact. 2013. Saatavissa: http://msdn.microsoft.com/en-us/library/ms172448.aspx. Hakupäivä 26.3.2013.

37. 101 LINQ Samples. 2013. Saatavissa: http://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b. Hakupäivä 9.4.2013.

LIITTEET

Liite 1 Lähtötietomuistio

Liite 2 Koodi: SQLite-tietokannan luonti

Liite 3 Koodi: Taulun luonti tietokantaan

Liite 4: Koodi: Mittauksien lisääminen tietokantaan

Liite 5: Koodi: Monen rivin hakeminen graafia varten

Liite 6: Koodi: DataSetin purkaminen tavallisiin taulukoihin

Liite 7: Koodi: ZedGraph-graafin piirtäminen

Liite 8: Scrumin excel-dokumentaation sivut kuvina

LÄHTÖTIETOMUISTIO

Tekijä Miika Kontio,

Tilaaja Oy PremiSense Ltd.

Tilaajan yhdyshenkilö ja yhteystiedot Jussi Vepsäläinen,

Työn nimi Opinnäytetyö. Tietokanta sensoridatalle, ja datan jatkoanalysointi

Työn kuvaus Tietokannan määrittely, suunnittelu, toteutusta. Opinnäytetyön teoriaosuudessa kerrotaan aluksi tietokannoista yleensä. Sitten kerrotaan miksi valittiin juuri se tietokanta mikä valittiin, ja verrataan sitä muihin tietokantoihin. Työhön kuuluu myös Scrum-projektimalli ja dokumentointi. Tietokannan on täytettävä tilaajan vaatimukset.

Työn tavoitteet Saada sensoridataa varten toimiva tietokanta määriteltyä, suunniteltua ja mahdollisesti myös toteutettua ainakin mahdollisimman pitkälle. Lisätavoitteena on työskentely yhdessä UI:n ja laitteistopuolen ihmisten kanssa, jotta tietokanta on UI:n ja laitteiston kanssa yhteensopiva.

Tavoiteaikataulu Projektin Scrumin Sprint 0 aloitettiin firman tiloissa 10.12.2012. Sprint 0:ssa tutustutaan ja perehdytään työhön ja tietokantoihin. Sen sovittiin päättyvän tammikuulla.Tämän jälkeen työn toteutusvaihe alkaa. Tekemiselle varattuna n. 250 tuntia ja työn teoriaosuudelle eli kirjoittamiselle on varattu loput n. 150 tuntia. Työn kesto yhteensä näin ollen n. 400 tuntia opinnäytetyön ohjeen mukaisesti. Tarkemmat aikataulutukset ja suunnitelmat tulevat projektin Scrum-dokumenttiin ja opinnäytetyön nettisivulle. Työn tulee valmistua maaliskuun lopulle.

```
//SQLite-tiedoston luonti. Ei tarkoitettu käytettäväksi monta kertaa ohjelman
//suorituksen aikana
public void CreateSQLiteFile(string fileName)
{
   //Tutkitaan onko tiedostoa vielä olemassa, jos ei, jatketaan
   if (!System.IO.File.Exists(fileName))
   {
      //Luo tyhjän sqlite-tiedoston
      SQLiteConnection.CreateFile(fileName);
      //luo yhteyden. Oma funktio.
      SetConnection(fileName);
      //Avaa yhteyden
      conn.Open();
      //Luo komennon jolla laitetaan viiteavaimet toimintaan
      SQLiteCommand cmd = new SQLiteCommand("PRAGMA foreign_keys = ON", conn);
      //Toteuttaa komennon
      cmd.ExecuteNonQuery();
      //Sulkee yhteyden
      conn.Close();
   }
   else
   {
      MessageBox.Show("Database file already exists. Cancelling..");
   }
}

//Funktio jolla luodaan yhteys SQLiteen erilaisin asetuksin
private void SetConnection(string filename)
{
   conn = new SQLiteConnection("Data Source=" + filename +
   ";Version=3;New=False;PRAGMA cache_size=20000;PRAGMA page_size=32768,PRAGMA
   synchronous=off;PRAGMA journal_mode=off");
}
```

```
public void CreateTable(string tablename)
{
   //merkkijonomuuttuja komentoa varten
   string sqlQuery = "";
   //otetaan yhteys SQLiteen
   SetConnection();


   /*komentoa varten SQL-lause joka luo taulun sensoreille. Olen käyttänyt
VARCHAR-tyyppejä, vaikka SQLite tunnistaa ne TEXT-tyypeiksi, koska näin lause
toimii suoraan myös MySQL-tietokannassa. SQL-lauseessa luodaan tässä myös
viiteavaimet*/
   sqlQuery = "CREATE TABLE IF NOT EXISTS sensorSettings (IDsensorSettings INTE-
GER PRIMARY KEY AUTOINCREMENT, MacAddress VARCHAR(20), ShortNodeID VARCHAR(4),
Nickname VARCHAR(20) UNIQUE, State VARCHAR(10), ErrorLogText TEXT, RoomID INT NOT
NULL, PulseRate INT, SensorType VARCHAR(4), CONSTRAINT RoomID FOREIGN KEY(RoomID)
REFERENCES room(IDroom)) ;";


   //try-lohko yrittää luoda komennon, avata yhteyden ja toteuttaa komennon
   try
   {
      //luodaan komento, avataan yhteys ja toteutetaan komento
      command = conn.CreateCommand();
      command.CommandText = sqlQuery;
      command.CommandType = CommandType.Text;
      conn.Open();
      command.ExecuteNonQuery();
   }
   /*jos try-lohkon sisältö epäonnistuu, kutsutaan catchia jossa kerrotaan
SQLite-virheilmoitus käyttäjälle MessageBoxilla eli viesti-ikkunalla*/
   catch (SQLiteException ex)
   {
      MessageBox.Show("Taulun luonti ei onnistu! " + ex.ToString());
   }
   /*suoritettiinpa sitten try tai catch, lopulta ajetaan finally-lohko jossa
suljetaan yhteys SQLiteen jos yhteys on olemassa*/
   finally
   {
      if (conn != null)
      conn.Close();
   }
}
```

```
//Datan simulointia tai muuten vaan massiivista rivimäärän lisäämistä varten
tehty funktio, jota kutsutaan StartTheThread-funktiosta säikeellä
public void WorkerThreadFunctionInsert(object dataTypeNameParam)
{
   //Muteksin lukitseminen
   myCopyOfMutex.WaitOne();

   //Muutetaan dataTypeNameParam-parametri objektista stringiksi ja asetetaan
     string-muuttujaan dataTypeName
   string dataTypeName = (string)dataTypeNameParam;

   //Ajastimen nollaus ja uudelleenkäynnistys tämän hakutapahtuman ajanottoa
     varten
   timer.Reset();
   timer.Start();

   //muistutetaan sqliteä että tiedostonimi on oikea
    sqlite.Filename = "HomeMonitorDatabase.sqlite";

   //lisättävien rivien määrä, tulee muuttujasta joka on ylempänä määritelty
   int rowCount = yleinenRowCount;

   //Otetaan DateTime-luokan avulla staattisesti nykyhetki talteen now-muuttujaan
   DateTime now = DateTime.Now;
   //muutetaan aika string-muotoon oikealla formaatilla
   string aika = now.ToString(@formaatti);

   //luodaan taulukko johon sijoitetaan tarpeelliset arvot dataMeasurement-
   //tauluun rivin lisäystä varten
   string[] array = new string[5];

   //luodaan tyhjä string-muuttuja datatyypin ID:tä varten
   string dataTypeID = "";

   //luodaan tyhjä DataSet hakutuloksen sijoitusta varten
   DataSet ds = new DataSet();

   //tutkitaan parametriä ja haetaan sen mukaan dataTyypille kaikki tiedot.
   //oikeassa koodissa tässä on else if-lauseita monille eri datatyypin nimille.
   if (dataTypeName == "temperature")
   {
      ds = sqlite.GetDataSetWithQuery("SELECT * FROM dataType WHERE DataTypeName
                                       = 'temperature' ;");
   }
   //otetaan DataSetistä taulukko DataTable-tyyppiseen muuttuujaan dt
```

```
    DataTable dt = ds.Tables[0];

    //käydään dt läpi foreach-luupilla rivi kerrallaan ja otetaan IddataType-
    //talteen.
    //Periaatteessa pitäis olla vain yks rivi koska dataType-taulussa on vain yksi
    //rivi per tietty DataTypeName eli edellisestä hausta pitäisi olla tullut vain
    //1 rivi tuloksia.
    foreach (DataRow row in dt.Rows)
    {
        dataTypeID = row["IDdataType"].ToString();
    }

    //sijoitetaan taulukkoon kovakoodattuja arvoja null sekä sensorin ID ja
    //ei-kovakoodattuja arvoja dataTypeID ja aika, sekä tyhjä dataValue, joka
    //luodaan oikeasti vasta SQLiteController.cs-luokassa InsertManyRows-metodissa
    array[0] = "null"; //ID:ksi null
    array[1] = "1"; //sensorSettingsID eli sensorin ID
    array[2] = dataTypeID; //haetun datatyypin ID
    array[3] = aika; //MeasureDatetime eli lisäysaika
    array[4] = ""; //DataValue eli mittaustulos asetetaan SQLiteController.cs-
                   //luokassa

    if (dataTypeName == "temperature")
    {
        //simulointifunktion kutsu lämpötilamittausrivien lisäyksille sensorille 1.
          Siellä tapahtuu vasta varsinainen SQLite-tietokantaan lisäys.
        sqlite.InsertManyRowsToDataMeasurementTable("dataMeasurement",
                                                    "temperature", array,
                                                     rowCount);
    }

    //ajastimen pysäytys ja tähän funktioon kuluneen ajan tulostus debug-ikkunaan
    timer.Stop();
    Debug.WriteLine("Time Taken: " +
    timer.Elapsed.TotalMilliseconds.ToString("#,##0.00 'milliseconds'"));

    //Muteksin vapautus
    myCopyOfMutex.ReleaseMutex();
}
```

Varsinainen rivien lisääminen tapahtuu SQLiteController.cs-luokassa funktiossa InsertManyRowsToDataMeasurementTable. Tässä esimerkkinä usean rivin lisäämisestä käytetään satunnaisten simuloitujen lämpötilamittausten lisäämistä. Tietoa lisätään dataMeasurement-tauluun tietokantaan.

```
//Monen rivin lisäämistä varten luotu funktio
public void InsertManyRowsToDataMeasurementTable(string tableName, string
dataType, string[] paramArray, int howMany)
{
   //purkkaratkaisuna varmistetaan tässä oikea filename
   filename = "HomeMonitorDatabase.sqlite";
   //yhteyden luonti sqliteen
   SetConnection();

   //INSERT-lause jolla lisätään hetken päästä rivejä dataMeasurement-tauluun
   //sqlite-tietokantaan
   string sqlQueryDataMeasurement = @"INSERT INTO " + tableName +
   " VALUES($paramIDDataMeasurement, $paramSensorSettingsID, $paramDataTypeID,
$paramMeasureDatetime, $paramDataValue);";

   //sqliten komennon luonti
   command = conn.CreateCommand();

   //komennolle annetaan parametreiksi null sekä saapuvan string taulun jäseniä
   command.Parameters.AddWithValue("$paramIDDataMeasurement", null);
   command.Parameters.AddWithValue("$paramSensorSettingsID", paramArray[1]);
   command.Parameters.AddWithValue("$paramDataTypeID", paramArray[2]);
   command.Parameters.AddWithValue("$paramMeasureDatetime", paramArray[3]);
   command.Parameters.AddWithValue("$paramDataValue", paramArray[4]);

   //komennon tekstiksi annetaan hetki sitten luotu tietokantalause
   command.CommandText = sqlQueryDataMeasurement;

   //satunnaista mittaustulosta varten luodaan Random-luokasta muuttuja
   Random randomDouble = new Random();

   //muuttujat arvontaa varten, minimi ja maksimi
   double max = 0.0;
   double min = 0.0;

   //Annetaan eri datatyypeille eri minimi- ja maksimiarvot
   if (dataType == "temperature")
   {
      max = 25.0;
      min = 17.0;
   }

   //Mittaustuloksien DataValueita varten uusi double-taulukko
   double[] dataValue = new double[howMany];
```

```
//arvotaan niin monta randomoitua mittaustuloksen dataValueta kuin on
//lisättäviä rivejä
for (int i = 0; i < howMany; i++)
{
   //arvoaan luku minimin ja maksimin väliltä 2 desimaalilla taulukkoon
   dataValue[i] = Math.Round(randomDouble.NextDouble() * (max - min) + min,
   2);
}

//sqliten käyttämä formaatti datetime-tyyppisille sarakkeille
string formatOma = @"yyyy-MM-dd HH:mm:ss";

//nykyaika talteen
DateTime now = DateTime.Now;
//ensimmäisen simudatan aika talteen
DateTime someTimeAgo = now.AddMinutes(-howMany);

//yritetään tehdä tietokantajutut try-lohkossa
try
{
   //yhteden avaaminen sqliteen
   conn.Open();
   //transaktion eli käskyjoukon aloitus. Tärkeä!
   mytransaction = conn.BeginTransaction();

   //tehdään for-luupin sisällä tietokantakäskyjä
   for (int n = 0; n < howMany; n++)
   {
      //lisätään ensimmäiseen aikaan aina kierroslaskurin arvo minuutteina
      DateTime timeForDatabase = someTimeAgo.AddMinutes(n);

      //muutetaan aika stringiksi oikeaan formaattiin
      string aika = timeForDatabase.ToString(formatOma);

      //Muutetaan simulointia varten mittausaikaa ja mittausarvoa jokaisella
      //luupin kierroksella, että saadaan tietokantaan eri arvoilla ja eri
      //ajoilla mitattua dataa
      command.Parameters["$paramMeasureDatetime"].Value = aika;
      //Arvo pitää muuttaa doublesta decimal-tyyppiseksi että se menee
      //sqliteen oikein
      command.Parameters["$paramDataValue"].Value =
             Convert.ToDecimal(dataValue[n]);

      //komennon suorituskäsky
      command.ExecuteNonQuery();
```

```
            }
        }
        //jos try-lohko epäonnistuu, heitetään messageboxi, jossa lukee virheilmoitus
        catch (SQLiteException ex)
        {
            MessageBox.Show("VIRHE! " + ex.ToString());
        }
        //lopuksi tehdään tämä aina
        finally
        {
        //committoi transaktio eli tee koko siirto-operaatio kaikkine käskyineen, jos
        //transaktio-olio on olemassa
            if(mytransaction != null)
                mytransaction.Commit();

            //jos yhteys on olemassa, sulje se ja vapauta sqliten lukitus
            if (conn != null)
                conn.Close();
        }
    }
```

```
public void ThreadSelectDataMeasurements(object dataTypeName)
{
   //Muteksin lukitseminen
   myCopyOfMutex.WaitOne();

   //ajastin, joka mittaa tähän funktioon kulunutta suoritusaikaa, päälle
   timer.Reset();
   timer.Start();

   sqlite.Filename = "HomeMonitorDatabase.sqlite";
   sqlite.TableName = "dataMeasurement";

   //vaihdetaan aika sqliten tukemaan formaattiin
   string formatOma = @"yyyy-MM-dd HH:mm:ss";

   string aika = "";
   string dataTypeForQuery = "";

   //tyhjennetään DataSet-lista jossa on viimeiset hakutulokset tallessa
   oMain.dataSetForGraphList.Clear();

   string query = "";
   string groupByString = "";
   //object-tyypistä stringiin
   string dataType = (string)dataTypeName;


   if (dataType == "airQuality")
   {
      //datetimepicker-kontrollista valittu ajankohta
      aika = Main.indoorPickedDateTime;

      if (Main.selectedIndoorDataType == "Lämpötila")
      {
         dataTypeForQuery = "temperature";
      }
      //tässä on todellisuudessa else if-haaroja eri datatyypeille, mutta ne
      //eivät ole työn kannalta oleellisia

      //Pilkotaan aikaleima joka on tyyppiä "yyyy-MM-dd HH:mm:ss";
      //vuosiksi,kuukausiksi,päiviksi,tunneiksi,minuuteiksi ja sekunneiksi ja
      //muutetaan ne int muotoon, jotta voidaan luoda DateTime pastTime
      string years = aika.Substring(0, 4);
      string months = aika.Substring(5, 2);
      string days = aika.Substring(8, 2);
      string hours = aika.Substring(11, 2);
      string minutes = aika.Substring(14, 2);
      string seconds = aika.Substring(17, 2);

      int iyears = 0;
      int imonths = 0;
      int idays = 0;
      int ihours = 0;
      int iminutes = 0;
      int iseconds = 0;

      int.TryParse(years, out iyears);
      int.TryParse(months, out imonths);
      int.TryParse(days, out idays);
      int.TryParse(hours, out ihours);
      int.TryParse(minutes, out iminutes);
      int.TryParse(seconds, out iseconds);
```

```
//otetaan nykyaika
DateTime now = DateTime.Now;

//Luodaan muuttuja menneelle ajalle
DateTime pastTime = new DateTime(iyears, imonths, idays, ihours, iminutes,
                                 iseconds);

//switch case haara eri aikavälivalinnoille
switch (oMain.currentGraphSettingMain)
{
   case GraphSettingsEnum.Hour:
      pastTime = pastTime.AddMinutes(-60);
      groupByString = "%M"; //min
   break;
   case GraphSettingsEnum.Day:
      pastTime = pastTime.AddHours(-24);
      groupByString = "%H"; //hour
   break;
   case GraphSettingsEnum.Week:
      pastTime = pastTime.AddDays(-7);
      groupByString = "%d"; //day
   break;
   case GraphSettingsEnum.Month:
      pastTime = pastTime.AddMonths(-1);
      groupByString = "%d"; //day
   break;
   case GraphSettingsEnum.Year:
      pastTime = pastTime.AddYears(-1);
      groupByString = "%m"; //month
   break;
}

string mennytaika = pastTime.ToString(formatOma);

DataSet ds = new DataSet();
ViewEnum currentView = oMain.CurrentView;

//tutkitaan ollaanko
//ilmanlaatu-näkymässä
if (currentView == ViewEnum.AirQuality)
{
   //Haetaan room-taulusta IDroom, koska sitä tarvitaan dataMeasurement-
   //taulusta tietojen hakuun
   ds = sqlite.GetDataSetWithQuery("SELECT * FROM room WHERE roomName = '"
        + Main.selectedIndoorRoom + "' ;");
}

//tarkastetaan löytyikö äskeisellä haulla tietoja eli onko dataSet tyhjä
//vai ei
bool RoomIsEmpty = sqlite.IsEmpty(ds);

if (RoomIsEmpty == false)
{
   DataTable dt = ds.Tables[0];

   string roomID = dt.Rows[0]["IDroom"].ToString();

   //Haetaan sensorSettings-taulusta kaikki Nicknamet (jotka erottavat
   //uniikisti mistä tietueesta on kyse)
   DataSet ds2 =
     sqlite.GetDataSetWithQuery("SELECT Nickname FROM sensorSettings;");
```

```
        bool sensorIsEmpty = sqlite.IsEmpty(ds2);

        if(sensorIsEmpty == false && aika != "")
        {
           //Nicknameja varten dynaaminen lista
           List<string> stringList = new List<string>();

           //käydään dataset läpi taulu kerrallaan
           foreach (DataTable dtTable in ds2.Tables)
           {
              //käydään taulut läpi rivi kerrallaan
              foreach (DataRow dataRow in dtTable.Rows)
              {
                 string s = dataRow["Nickname"].ToString();
                 //..ja lisätään nicknamet listaan
                 stringList.Add(s);
              }
           }

    for (int j = 0; j < stringList.Count; j++)
    {
    //iso SQL-hakulause
    query = "Select datat.MeasureDatetime, AVG(DataValue) AS AverageData,
            sensorit.Nickname " +
            "FROM dataMeasurement AS datat " +
            "JOIN dataType AS tyypit ON datat.DataTypeID = tyypit.IDdataType " +
            "JOIN sensorSettings AS sensorit ON datat.SensorSettingsID = senso
             rit.IDsensorSettings " +
            "JOIN room AS huoneet ON sensorit.RoomID = huoneet.IDroom " +
            "WHERE datat.MeasureDatetime BETWEEN " + "'" + mennytaika + "' AND " +
            "'" + aika + "' " +
            "AND tyypit.DataTypeName = '" + dataTypeForQuery + "' " +
            "AND huoneet.IDroom = " + roomID + " " +
            "AND sensorit.Nickname = '" + stringList[j] + "' " +
            "GROUP BY strftime('" + groupByString + "', datat.MeasureDatetime) OR
             DER BY datat.IDdataMeasurement DESC;";

            //talletetaan haetut tulokset dataSettiin
             DataSet dsForList = sqlite.GetDataSetWithQuery(query);

            //tarkistetaan onko dataSetissä tuloksia
             bool boolean = sqlite.IsEmpty(dsForList);

             //lisätään ko. DataSet DataSet-listaan Main-luokassa
             if (boolean == false)
                oMain.dataSetForGraphList.Add(dsForList);
             }
    }
}

//Pysäytetään ajastin ja debugataan kauanko ajastin mittasi aikaa
timer.Stop();
Debug.WriteLine("Time taken for select: " + tim
                er.ElapsedMilliseconds.ToString("#,##0.00 'milliseconds'"));

//lopulta lähetetään säikeeltä mainille viesti delegaatilla
//viesti kertoo että säie on valmis ja parametrina lähetetään mittaustyyppi
//(esim lämpötila)
oMain.Invoke(oMain.m_DelegateThreadFinishedSelecting, dataTypeForQuery);
```

```
    // Muteksin vapautus
    myCopyOfMutex.ReleaseMutex();

}
```

```
//DataSetin purkaminen tavallisiin taulukoihin
public void chartViewIndoor(List<DataSet> dataSetList, string dataType,
                            GraphSettingsEnum current)
{
   //graafi asetukset (aikaväli)
   currentGraphSettings = current;

   // Tyhjennetään zedgraph-taulukko
   zgcIndoorChart.GraphPane.CurveList.Clear();

   //otetaan talteen kuinka monta DataSettiä löytyi haulla (eli kuinka monelta
   //eri sensori+sensorityyppi-yhdistelmältä)
   int listcount = dataSetList.Count;

   //koitetaan laittaa graafille vain jos datasetlistalla on enemmän kuin 0
   //datasettiä

   if (listcount > 0)
   {
      //pyöritään dataSetLista läpi ja piirretään löytyneet datasetit graafille.
      //Yksi dataSet vastaa yhden sensorin lukemia valitussa huoneessa
      for (int i = 0; i < listcount; i++)
      {
         string[] timeArray = dataHelper.GetTimeFromDataSet(dataSetList[i]);
         double[] dataArray = dataHelper.GetDataFromDataSet(dataSetList[i]);

         //funktio jossa piirretään x- ja y-akselin mukaisia juttuja ZedGraphiin
         IndoorDataToGraph(i, timeArray, dataArray, dataType);
      }
   }
   else
   {
      MessageBox.Show("No data available. ei graafia kiitos..");
   }

//funktio, joka käy läpi DataSetin ja palauttaa aikaleimat string-tauluna
public string[] GetTimeFromDataSet(DataSet dataSet)
{
   //Jos dataSetissa on rivejä..
   if (dataSet.Tables[0].Rows.Count > 0)
   {
      //DataSetistä DataTableksi
      DataTable dt = dataSet.Tables[0];
      //Dynaaminen List-objekti string-aikoja varten
      List<string> timeList = new List<string>();
```

```
//käydään läpi kaikki Datarivit DataTablessa
        foreach (DataRow dr in dt.Rows)
        {
            //Otetaan datariviltä datetime-aikaleima talteen
            string aika = dr["MeasureDatetime"].ToString();
            int aikaLen = aika.Length;
            //otetaan aikaleimasta turhat sekunnit pois substringillä ja lisätään
            //timeList-listalle
            timeList.Add(aika.Substring(0, aikaLen - 3));
        }

        //otetaan listan jäsenten määrä talteen
        int listCount = timeList.Count;
        //Luodaan tavallinen taulukko joka on yhtä pitkä kuin listCount
        string[] timeArray = new string[listCount];

        //listalta arrayhyn
        for (int j = 0; j < listCount; j++)
        {
            timeArray[j] = timeList[j];
        }

        //palautetaan string taulukko funktionkutsupaikkaan
        return timeArray;
    }
    //Jos DataSetissä ei ole rivejä
    else
    {
        string[] array = new string[0];
        return array;
    }
}
```

```
//ZedGraph-graafin piirtäminen tehdään tällä funktiolla

private void IndoorDataToGraph(int index, string[] timeArray, double[] dataArray,
                               string dataType)
{
   //Luodaan viiva graafia varten
   LineItem lineItem = new LineItem("");

   //jos dataa löytyy enemmän kuin 0 riviä jatketaan
   if (dataArray.Length > 0)
   {
      //Otetaan suurin ja pienin arvo dataArraysta talteen
      double max = dataArray.Max();
      double min = dataArray.Min();

      //Luodaan double-tyyppinen lista johon päivämäärät kohta sijoitetaan
      List<double> DatesX = new List<double>();

      //Luodaan kaksi DateTime-muuttujaa suurimmalle ja pienimmälle arvolle
      DateTime Max = new DateTime();
      DateTime Min = DateTime.Now;

      //muutetaan DateTimet XDateksi ja siitä doubleksi, sekä tutkitaan Max ja
      //Käydään timeArray-taulukko läpi luupilla
      for (int i = 0; i < timeArray.Length; i++)
      {
         //Otetaan taulukon joka kierros löydetty DateTime talteen muuttujaan
         DateTime datetime = (Convert.ToDateTime(timeArray[i]));

         //Tutkitaan joka kierros onko kyseisen kierroksen DateTime suurempi kuin
         //Max-muuttujan sisältö. Jos on,
         //niin vaihdetaan Max-muuttujan arvoksi tämä
         //uusi isompi päivämäärä
         if (datetime > Max)
          {
             Max = datetime;
          }

         //Tutkitaan joka kierros onko kyseisen kierroksen DateTime pienempi kuin
         //Min-muuttujan sisältö. Jos on,
         //niin vaihdetaan Min-muuttujan arvoksi tämä
         //uusi pienempi päivämäärä

          if (datetime < Min)
          {
```

```
            Min = datetime;
}

        //Muutetaan DateTime XDateksi ja otetaan talteen xDate-muuttujaan
        XDate xDate = dataHelper.ConvertDateToXdate(datetime);
        //Lisätään xDate-muuttuja doubleksi muutettuna DatesX-listalle
        DatesX.Add((double)xDate);
    }

    //Muutetaan max ja min DateTimet myös XDateksi ZedGraphia varten.
    XDate ekaXDate = dataHelper.ConvertDateToXdate(Min);
    XDate viimeinenXDate = dataHelper.ConvertDateToXdate(Max);

    //Tästä alkaa ZedGraphin asetusten laittaminen:

    //Fontin skaalaus pois päältä
    paneIndoor.IsFontsScaled = false;
    //Fontin koon vaihto otsikossa aiemmin määritellyn muuttujan arvon mukaan
    paneIndoor.Title.FontSpec.Size = paneTitleSize;

    //Vaihdetaan ZedGraphin otsikon teksti datatyypin mukaan
    if (dataType == "temperature") { paneIndoor.Title.Text = "Lämpötila (" +
        dataHelper.degree + " C)"; }
    //tässä on else if-lohkoja sitten myös eri datatyypeille, mutta en näytä
      niitä tässä työssä koska se ei ole olennaista

    //X-akselin arvojen muutos aikavälin ”Vuosi” mukaan
    if (currentGraphSettings == GraphSettingsEnum.Year)
    {
       paneIndoor.XAxis.Scale.Format = "MM.yyyy";
       paneIndoor.XAxis.Type = AxisType.Date;
       paneIndoor.XAxis.Scale.MajorUnit = DateUnit.Month;
       paneIndoor.XAxis.Scale.MajorStep = 1;
    }
    //X-akselin arvojen muutos aikavälin ”Kuukausi” mukaan
    else if (currentGraphSettings == GraphSettingsEnum.Month)
    {
       paneIndoor.XAxis.Scale.Format = "dd.MM";
       paneIndoor.XAxis.Type = AxisType.Date;
       paneIndoor.XAxis.Scale.MajorUnit = DateUnit.Day;
       paneIndoor.XAxis.Scale.MajorStep = 1;
    }
    //tässä on else if-lohkoja sitten myös aikaväleille viikko jne. ,
    //mutta en näytä
```

```
        //niitä tässä työssä koska se ei ole olennaista

        //X-akselin fontin koon muutos
        paneIndoor.XAxis.Scale.FontSpec.Size = paneXaxisFontSizeMax;
        //X-akselin otsikon piilotus
        paneIndoor.XAxis.Title.IsVisible = false;
        //X-akselin apuviivojen piirto päälle
        paneIndoor.XAxis.MajorGrid.IsVisible = true;
        //X-akselin skaalaus pienimmän ja suurimman löydetyn ajan mukaan
        paneIndoor.XAxis.Scale.Min = ekaXDate;
        paneIndoor.XAxis.Scale.Max = viimeinenXDate;
        //Y-akselille vastaavia asetuksia kuin x-akselille
        paneIndoor.YAxis.Scale.FontSpec.Size = paneYaxisFontSizeMax;
        paneIndoor.YAxis.Scale.Max = max + 0.5;
        paneIndoor.YAxis.Scale.Min = min - 0.5;
        paneIndoor.YAxis.Title.IsVisible = false;
        paneIndoor.YAxis.MajorGrid.IsVisible = true;
        //Viivan piirto ZedGraphiin
        lineItem = paneIndoor.AddCurve("Lämpötila", DatesX.ToArray(), dataArray,
                    Color.Black, SymbolType.None);
        //Viivan paksuuden muutos ja pehmennys
        lineItem.Line.Width = 3.0F;
        lineItem.Line.IsSmooth = true;
        lineItem.Line.SmoothTension = 0.3F;
        lineItem.Label.IsVisible = false;
        //ZedGraphin piirtokutsut
        zgcIndoorChart.AxisChange();
        zgcIndoorChart.Invalidate();
    }
}
```

YRITYKSEN NIMI	Oy PremiSense Ltd
PROJEKTIN NIMI	Opinnäytetyö: tietokannan määrittely, suunnittelu ja toteutusta sensoridatalle
SCRUM MASTER	Jussi Vepsäläinen
PROJEKTIN AIKATAULU	10.12.2012 – 22.2.2013 + kirjoituosa 25.2.2013 - 29.3.2013

DOKUMENTIN NIMI	scrum_seuranta_miika_kontio_UUSI.xls
TAULUKON NIMI	Resurssien käytettävyys

koulu

Henkilö	ARKIPÄIVIÄ PROJEKTIN AIKANA	MUUT PROJEKTIT	LOMAT	MUUT	PROJEKTITYÖN TEKEMISEEN PÄIVIÄ	PROJEKTIYÖN TEKEMISEEN TUNTEJA
Miika Kontio	58,0	0,0	10,0	8,0	40,0	280,0
					0,0	0,0
					0,0	0,0
					0,0	0,0
Yhteensä	**58,0**	**0,0**	**10,0**	**8,0**	**40,0**	**280,0**

Tavallinen oppari	TUNTEJA	PÄIVIÄ (tunnit / 7)
Koko opinnäytetyön kesto	405	58
Tekstiosuus	150	21
Toteutusosuus	255	36

Kuinka toteutin opparin	TUNTEJA	PÄIVIÄ (tunnit / 7)	
Koko opinnäytetyön kesto	405	58	
Tekstiosuus	125	18	Vähensin tekstiosaa 3 päivää!
Toteutusosuus	280	40	Lisäsin toteutusosaan 4 päivää!
	yhteensä		

Paljonko työpäiviä lopulta meni opparii:	PÄIVIÄ KULUI	
Toteutusosa	40	
Tekstiosa	25	Tekstiosuus meni 7 päivää yliajalle
Yhteensä	65	Koko oppariin kului 7 päivää enemmän kuin suunniteltiin
Entä tunteja:	TUNTEJA KULUI	
Toteutusosa	282	
Tekstiosa		
Yhteensä	282	

Resurssien_käytettävyys | Julkaisusuunnitelma | Kaikki tuotteen ominaisuudet | Tuotteen ominaisuuslista | Sprintin 0 tehtävälista | Sprintin 1 tehtävälista

PROJEKTIN NIMI	Opinnäytetyö: tietokannan määrittely, suunnittelu ja toteutusta sensoridatalle
SCRUM MASTER	Jussi Vepsäläinen
PROJEKTIN AIKATAULU	10.12.2012 – 22.2.2013 + kirjoituosa 25.2.2013 - 29.3.2013

DOKUMENTIN NIMI	scrum_seuranta_miika_kontio_UUSI.xls
TAULUKON NIMI	Julkaisusuunnitelma

PROJEKTISSA JÄSENIÄ	1
SPRINTTEJÄ PROJEKTISSA	5
TUNTEJA PER PÄIVÄ	7

PROJEKTIN RESURSSIT	10.12.2012 – 22.2.2013		
HENKILÖT	**KÄYTETTÄVISSÄ VIIKKOJA**	**KÄYTETTÄVISSÄ PÄIVIÄ**	**KÄYTETTÄVISSÄ TUNTEJA**
Miika Kontio	15	40	280
YHTEENSÄ		40	280

SPRINTIN 0 RESURSSIT	10.12.2012 – 4.1.2012			
HENKILÖ	**VIIKKOJA SPRINTISSÄ**	**PÄIVIÄ SPRINTISSÄ**	**TUNTEJA SPRINTISSÄ**	**OMINAISUUKSIA SPRINTISSÄ**
Miika Kontio	4	12	84	6
YHTEENSÄ		12	84	6
SPRINTIN 0 MAALI:	*Aloituspalaveri, eri tietokantoihin tutustuminen, tietokannan valinta, perehtyminen, testiapplikaatioiden luominen tietokantojen testailuun, projektin www-sivut, dokumentaatiot*			

SPRINTIN 1 RESURSSIT	07.01.2013 – 25.1.2013			
HENKILÖ	**VIIKKOJA SPRINTISSÄ**	**PÄIVIÄ SPRINTISSÄ**	**TUNTEJA SPRINTISSÄ**	**OMINAISUUKSIA SPRINTISSÄ**
Miika Kontio	3	12	84	8
YHTEENSÄ		12	84	8
SPRINTIN 1 MAALI:	*Tietokannan vaatimuslista kirjotettuna, tietokannalla jonkinlainen mallikaavio valmis, huollon tietokanta toteutettuna, sqlite integroituna pääohjelmaan ja perusfunktiot kunnossa*			

SPRINTIN 2 RESURSSIT	28.1.2013 – 8.2.2013			
HENKILÖ	**VIIKKOJA SPRINTISSÄ**	**PÄIVIÄ SPRINTISSÄ**	**TUNTEJA SPRINTISSÄ**	**OMINAISUUKSIA SPRINTISSÄ**
Miika Kontio	2	8	56	2
YHTEENSÄ		8	56	2
SPRINTIN 2 MAALI:	*Sensoridataa alustavasti tietokannassa, sensoridatan haku aikaleimalla pääohjelmalle toiminnassa, tietokannan testaus*			

SPRINTIN 3 RESURSSIT	11.2.2013 – 22.2.2013			
HENKILÖ	**VIIKKOJA SPRINTISSÄ**	**PÄIVIÄ SPRINTISSÄ**	**TUNTEJA SPRINTISSÄ**	**OMINAISUUKSIA SPRINTISSÄ**
Miika Kontio	3	8	56	3
YHTEENSÄ		8	56	3
SPRINTIN 3 MAALI:	*Valmis hyvin pääohjelman kanssa toimiva tietokanta, toimivat käyrien piirrot graafeissa*			

Resurssien_käytettävyys | Julkaisusuunnitelma | Kaikki tuotteen ominaisuudet | Tuotteen ominaisuuslista | Sprintin 0 tehtävälista | Sprintin 1 tehtävälista | Spri

YRITYKSEN NIMI	Oy PremiSense Ltd
PROJEKTIN NIMI	Opinnäytetyö: tietokannan määrittely, suunnittelu ja toteutusta sensoridatalle
SCRUM MASTER	Jussi Vepsäläinen
PROJEKTIN AIKATAULU	10.12.2012 – 22.2.2013 + kirjoituosa 25.2.2013 - 29.3.2013
DOKUMENTIN NIMI	scrum_seuranta_miika_kontio_UUSI.xls
TAULUKON NIMI	Kaikki tuotteen ominaisuudet

Sprintit värikoodattu helpottamaan tunnistamista

Ominaisuuden ID		arvosana: 1-5			
Korkea prioriteetti	**Ominaisuuden kuvaus, käyttäjätarina, toiminnallisuus**	**Story Point**	**Arvio toteutuksen kestosta (päivinä)**	**Sprintin numero**	**Huomioitavaa**
2	SQLite C# ohjelma	4	2	0	
3	Tietokannan valintakriteereihin tutustuminen ja tietokannan valinta	2	1	0	
4	Tietokannan suunnittelu	4	2	0	
7	SQLite-tietokannan suunnittelu	3	1	1	
8	SQLite-tietokantakomponentin funktiot	4	1	1	
9	SQLiten integrointi pääohjelmaan	2	1	1	
11	Sarjaportti-komponentista otettavien datojen laittaminen suoraan tietokantaan	1	1	1	
12	Graafin piirtäminen tietokannasta haetulla datalla	4	1	1, 2	Toteutettiin sprinttien 1 ja 2 aikana
13	XMLParser-komponentin luonti ja integrointi pääohjelmaan Talon perustietojen säilytystä varten	1	0,25	1	Tein tätä kotona pari tuntia
14	GIT versionhallinta	3		1	
15	Datan hakeminen ja käyttäminen	4	1	2	
16	UI toteutus ja julkisen tilan näyttö	3	1	2	
17	Graafin fixaus	4	1	3	
18	Julkisen tilan näyttö (public demo) ja siihen liittyvät asiat	4	1	3	
19	Visual studiosta deployment-paketin tekeminen sekä 32bit että 64bit järjestelmiin	2	1	3	

Matala prioriteetti	**Ominaisuuden kuvaus, käyttäjätarina, toiminnallisuus**	**Story Point**	**Arvio toteutuksen kestosta (päivinä)**	**Sprintin numero**	**Huomioitavaa**
1	MySQL C# ohjelma	3	0,5	0	
5	C# Chart-graafi	1	0,25	0	
6	Microsoft SQL Server CE C# ohjelma	1	0,25	0	
10	Datojen ottaminen tiedostosta tietokantaan	3	1	1, 2	Toteutettiin sprinttien 1 ja 2 aikana
Muu vaatimus	**Asiakas-/tuote-/järjestelmävaatimukset, laatuvaatimukset**				**Huomioitavaa**

Resurssien_käytettävyys | Julkaisusuunnitelma | **Kaikki tuotteen ominaisuudet** | Tuotteen ominaisuuslista | Sprintin 0 tehtävälista | Sprintin 1 tehtävälista | Spri

		Suunniteltu tuntimäärä	84	84	56	56			
		Sprintin numero	**1**	**2**	**3**	**4**			
Ominaisuuden ID	**Ominaisuuden kuvaus**	**Tunteja jäljellä projektissa**	280,0	201,0	117,0	57,5	-7,0		
		Sprintissä tunteja kului	79,0	84,0	59,5	64,5		**Huomioitavaa**	**Tila**
1	MySQL C# ohjelma		18,0						Valmis
2	SQLite C# ohjelma		14,5						Valmis
3	Tietokannan valintakriteereihin tutustuminen ja tietokannan valinta		14,0						Valmis
4	Tietokannan suunnittelu		11,0						Valmis
5	C# Chart-graafi		3,0						Valmis
6	Microsoft SQL Server CE C# ohjelma		3,0						Valmis
	Muut asiat		15,5						Valmis
	SPRINTIN 0 MAALI: Aloituspalaveri, eri tietokantoihin tutustuminen, tietokannan valinta testiapplikaatioiden luominen tietokantojen testailuun, projektin www-sivut, dokumentaatiot								
7	SQLite-tietokannan suunnittelu			11,5					Valmis
8	SQLite-tietokantakomponentin funktiot			26,5					Valmis
9	SQLiten integrointi pääohjelmaan			12,5					Valmis
10	Datojen ottaminen tiedostosta tietokantaan			0,0				Ominaisuus ID nro 10 siirretään Sprint 2:een	Ei aloitettu
11	Sarjaportti-komponentista otettavien datojen laittaminen suoraan tietokantaan			4,5					Valmis
12	Graafin piirtäminen tietokannasta haetulla datalla			4,0				Ominaisuus ID nro 12 siirretään Sprint 2:een	Valmis
13	XMLParser-komponentin luonti ja integrointi pääohjelmaan Talon perustietojen säilytystä varten			2,0					Valmis
14	GIT-versionhallinta			9,0					Valmis
	Muut asiat			14,0					Valmis
	SPRINTIN 1 MAALI: Tietokannan vaatimuslista kirjotettuna, tietokannalla jonkinlainen mallikaavio valmis, huollon tietokanta toteutettuna, sqlite integroituna pääohjelmaan ja perusfunktiot kunnossa								
10	Datojen ottaminen tiedostosta tietokantaan				0,0			Ominaisuus ID nro 10 jää toteuttamatta opparin aikana, koska se priorisoitiin ei-tärkeäksi	Ei aloitettu
12	Graafin piirtäminen tietokannasta haetulla datalla				14,5				Valmis
15	Datan hakeminen ja käyttäminen				15,5				Valmis
16	UI toteutus ja julkisen tilan näyttö				22,5				Valmis
	Muut asiat				7,0				Valmis
	SPRINTIN 2 MAALI: Sensoridataa alustavasti tietokannassa, sensoridatan haku aikaleimalla pääohjelmalle toiminnassa, tietokannan testaus								
17	*Graafin fixaus*					41,0			Valmis
18	*Julkisen tilan näyttö (public demo) ja siihen liittyvät asiat*					11,0			Valmis
	Visual studiosta deployment-paketin tekeminen sekä								

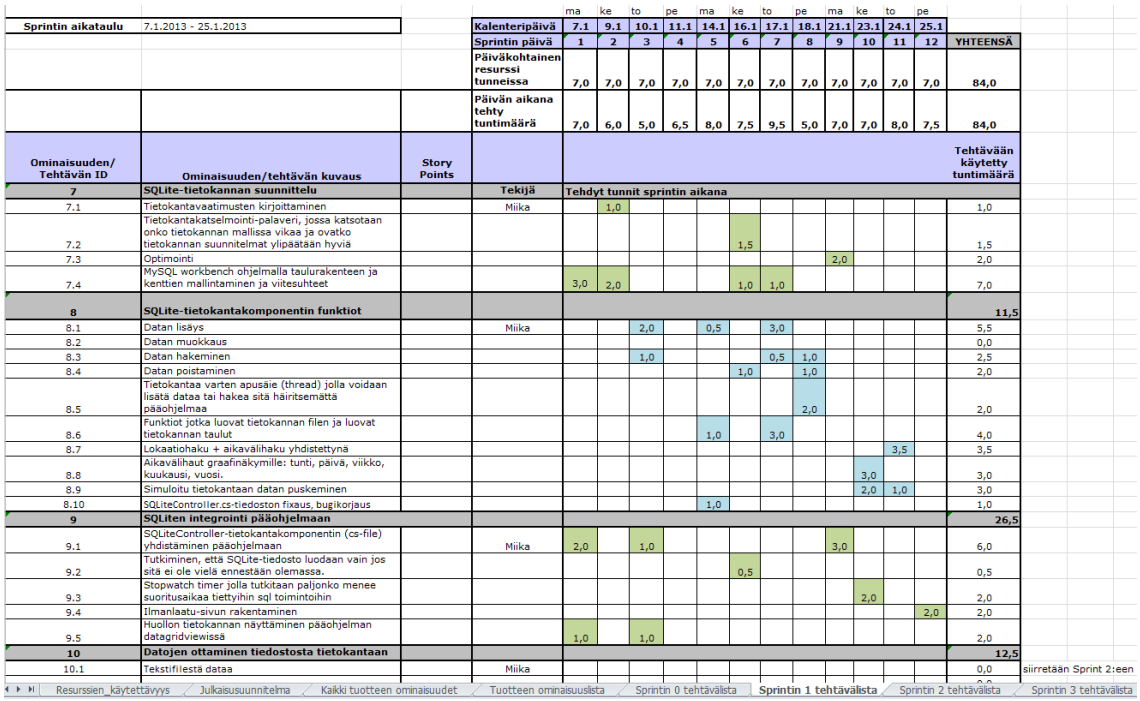
Tilakoodit
Valmis
Kesken
Ei aloitettu
Hylätty

Resurssien_käytettävyys | Julkaisusuunnitelma | Kaikki tuotteen ominaisuudet | **Tuotteen ominaisuuslista** | Sprintin 0 tehtävälista | Sprintin 1 tehtävälista | Sprintin 2 tehtävälista | Sprintin 3 tehtäv

	SPRINTIN 2 MAALI: Sensoridataa alustavasti tietokannassa, sensoridatan haku aikaleimalla pääohjelmalle toiminnassa, tietokannan testaus													
17	*Graafin fixaus*					41,0						Valmis		
18	*Julkisen tilan näyttö (public demo) ja siihen liittyvät asiat*					11,0						Valmis		
19	*Visual studiosta deployment-paketin tekeminen sekä 32bit että 64bit järjestelmiin*					12,0						Valmis		
	Muut asiat					0,5						Valmis		
	SPRINTIN 3 MAALI: Valmis hyvin pääohjelman kanssa toimiva tietokanta													

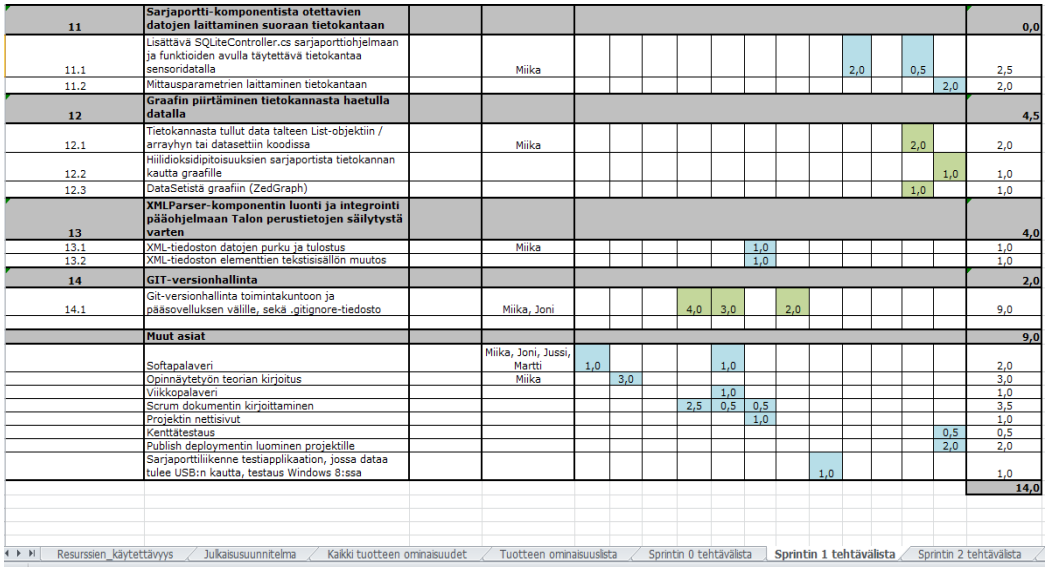
Sprintin aikataulu	10.12.2013 - 4.1.2013		**Kalenteripäivä**	**10.12**	**11.12**	**12.12**	**13.12**	**14.12**	**17.12**	**18.12**	**19.12**	**20.12**	**21.12**	**3.1**	**4.1**	
			Sprintin päivä	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**YHTEENSÄ**
			Päiväkohtainen resurssi tunneissa	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**84,0**
			Päivän aikana tehty tuntimäärä	**6,0**	**7,0**	**7,0**	**7,0**	**7,0**	**8,0**	**8,0**	**7,0**	**7,5**	**7,0**	**3,5**	**4,0**	**79,0**
Ominaisuuden/ Tehtävän ID	**Ominaisuuden/tehtävän kuvaus**	**Story Points**														**Tehtävään kägtetty tuntimäärä**
1	**MySQL C# ohjelma**		**Tekijä**	**Tehdyt tunnit sprintin aikana**												
1.1	XAMP MySQL serverin asennus ja toimintaan saattaminen		Miika			1,5										1,5
1.2	UI:n toteutus					2,0	3,0									5,0
1.3	C# MySQL komennot							4,0								4,0
1.4	Yhteyden otto C# ohjelman ja MySQL:n tietokannan välille					3,5	4,0									7,5
2	**SQLite C# ohjelma**															**18,0**
2.1	Tietokannan luonti ja siihen yhteyden ottaminen		Miika						1,0							1,0
2.2	Säie joka puskee tietokantaan testidataa									1,0	2,0					3,0
2.3	UI:n toteutus								2,0	1,0						3,0
2.4	SQL-lauseiden käyttö C# SQLitessä ja funktioiden toteutus									1,0	2,0	3,0				6,0
2.5	Huoltokirja-taulun ensimmäinen versio tietokantaan												1,5			1,5
3	**Tietokannan valintakriteereihin tutustuminen ja tietokannan valinta**															**14,5**
3.1	SQLiten dokumentaatioden lukeminen ja opiskelu		Miika						1,0				2,0			3,0
3.2	Netistä SQLiten käyttäjäkokemusten lukeminen foorumeilta								2,0			1,0				3,0
3.3	Yleinen tietokantoihin tutustuminen. Xampp serverin ja MySQL käytön opiskelua.				7,0							1,0				8,0
4	**Tietokannan suunnittelu**															**14,0**
4.1	Alustava tietokannan suunnittelu paperilla. Taulujen ja niiden kenttien määrät ja oletettu sisältö		Miika					2,0			2,0		0,5			4,5
4.2	Opparin pohja														2,0	2,0
4.2	Tietokannan spek.saus "power point" dokumenttiin							1,0			1,0	1,5	1,0			4,5
5	**C# Chart-graafi**															**11,0**
5.1	C# form-applikaatiossa graafin piirto		Miika							2,0			1,0			3,0
6	**Microsoft SQL Server CE C# ohjelma**															**3,0**
6.1	C# testiapplikaatio SQL Server CE:n testaukseen		Miika							3,0						3,0
	Muut asiat															**3,0**
	Viikkopalaveri		Miika, Jussi, Joni						1,0							1,0
	UI Design -palaveri		Miika, Jussi, Joni, Petri	3,0												3,0
	Opinnäytetön aloituspalaveri koululla		Miika, Jussi, Eero						1,0							1,0
	Tuntimäärien käytön suunnittelu (tuntiseuranta-lomake)		Miika									1,0	1,0			2,0
	Scrum dokumentaation kirjoittaminen		Miika											3,5	2,0	5,5
	Ohjelmistojen asennus: mm. Visual Studio 2012 ja ajurit		Miika	3,0												3,0
																15,5

Resurssien_käytettävyys | Julkaisusuunnitelma | Kaikki tuotteen ominaisuudet | Tuotteen ominaisuuslista | **Sprintin 0 tehtävälista** | Sprintin 1 tehtävälista | Spr



				ma	ke	to	pe	ma	ke	to	pe	ma	ke	to	pe	
Sprintin aikataulu	7.1.2013 - 25.1.2013		**Kalenteripäivä**	**7.1**	**9.1**	**10.1**	**11.1**	**14.1**	**16.1**	**17.1**	**18.1**	**21.1**	**23.1**	**24.1**	**25.1**	
			Sprintin päivä	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**YHTEENSÄ**
			Päiväkohtainen resurssi tunneissa	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**84,0**
			Päivän aikana tehty tuntimäärä	**7,0**	**6,0**	**5,0**	**6,5**	**8,0**	**7,5**	**9,5**	**5,0**	**7,0**	**7,0**	**8,0**	**7,5**	**84,0**
Ominaisuuden/ Tehtävän ID	**Ominaisuuden/tehtävän kuvaus**	**Story Points**														**Tehtävään käytetty tuntimäärä**
7	**SQLite-tietokannan suunnittelu**		**Tekijä**	**Tehdyt tunnit sprintin aikana**												
7.1	Tietokantavaatimusten kirjoittaminen		Miika		1,0											1,0
7.2	Tietokantakatselmointi-palaveri, jossa katsotaan onko tietokannan mallissa vikaa ja ovatko tietokannan suunnitelmat ylipäätään hyviä								1,5							1,5
7.3	Optimointi											2,0				2,0
7.4	MySQL workbench ohjelmalla taulurakenteen ja kenttien mallintaminen ja viitesuhteet			3,0	2,0				1,0	1,0						7,0
8	**SQLite-tietokantakomponentin funktiot**															**11,5**
8.1	Datan lisäys		Miika			2,0		0,5		3,0						5,5
8.2	Datan muokkaus															0,0
8.3	Datan hakeminen					1,0				0,5	1,0					2,5
8.4	Datan poistaminen								1,0		1,0					2,0
8.5	Tietokantaa varten apusäie (thread) jolla voidaan lisätä dataa tai hakea sitä häiritsemättä pääohjelmaa										2,0					2,0
8.6	Funktiot jotka luovat tietokannan filen ja luovat tietokannan taulut							1,0		3,0						4,0
8.7	Lokaatiohaku + aikavälihaku yhdistettynä													3,5		3,5
8.8	Aikavälihaut graafinäkymille: tunti, päivä, viikko, kuukausi, vuosi.												3,0			3,0
8.9	Simuloitu tietokantaan datan puskeminen												2,0	1,0		3,0
8.10	SQLiteController.cs-tiedoston fixaus, bugikorjaus							1,0								1,0
9	**SQLiten integrointi pääohjelmaan**															**26,5**
9.1	SQLiteController-tietokantakomponentin (cs-file) yhdistäminen pääohjelmaan		Miika	2,0		1,0						3,0				6,0
9.2	Tutkiminen, että SQLite-tiedosto luodaan vain jos sitä ei ole vielä ennestään olemassa.								0,5							0,5
9.3	Stopwatch timer jolla tutkitaan paljonko menee suoritusaikaa tiettyihin sql toimintoihin												2,0			2,0
9.4	Ilmanlaatu-sivun rakentaminen														2,0	2,0
9.5	Huollon tietokannan näyttäminen pääohjelman datagridviewissä			1,0		1,0										2,0
10	**Datojen ottaminen tiedostosta tietokantaan**															**12,5**
10.1	Tekstifilestä dataa		Miika													0,0 siirretään Sprint 2:een

Resurssien_käytettävyys | Julkaisusuunnitelma | Kaikki tuotteen ominaisuudet | Tuotteen ominaisuuslista | Sprintin 0 tehtävälista | **Sprintin 1 tehtävälista** | Sprintin 2 tehtävälista | Sprintin 3 tehtävälista



11	**Sarjaportti-komponentista otettavien datojen laittaminen suoraan tietokantaan**															**0,0**
11.1	Lisättävä SQLiteController.cs sarjaporttiohjelmaan ja funktioiden avulla täytettävä tietokantaa sensoridatalla		Miika									2,0		0,5		2,5
11.2	Mittausparametrien laittaminen tietokantaan														2,0	2,0
12	**Graafin piirtäminen tietokannasta haetulla datalla**															**4,5**
12.1	Tietokannasta tullut data talteen List-objektiin / arrayhyn tai datasettiin koodissa		Miika											2,0		2,0
12.2	Hiilidioksidipitoisuuksien sarjaportista tietokannan kautta graafille														1,0	1,0
12.3	DataSetistä graafiin (ZedGraph)													1,0		1,0
13	**XMLParser-komponentin luonti ja integrointi pääohjelmaan Talon perustietojen säilytystä varten**															**4,0**
13.1	XML-tiedoston datojen purku ja tulostus		Miika						1,0							1,0
13.2	XML-tiedoston elementtien tekstisisällön muutos								1,0							1,0
14	**GIT-versionhallinta**															**2,0**
14.1	Git-versionhallinta toimintakuntoon ja pääsovelluksen välille, sekä .gitignore-tiedosto		Miika, Joni				4,0	3,0		2,0						9,0
	Muut asiat															**9,0**
	Softapalaveri		Miika, Joni, Jussi, Martti	1,0				1,0								2,0
	Opinnäytetyön teorian kirjoitus		Miika		3,0											3,0
	Viikkopalaveri							1,0								1,0
	Scrum dokumentin kirjoittaminen						2,5	0,5	0,5							3,5
	Projektin nettisivut								1,0							1,0
	Kenttätestaus														0,5	0,5
	Publish deploymentin luominen projektille														2,0	2,0
	Sarjaporttiliikenne testiapplikaation, jossa dataa tulee USB:n kautta, testaus Windows 8:ssa										1,0					1,0
																14,0

Resurssien_käytettävyys | Julkaisusuunnitelma | Kaikki tuotteen ominaisuudet | Tuotteen ominaisuuslista | Sprintin 0 tehtävälista | **Sprintin 1 tehtävälista** | Sprintin 2 tehtävälista

SPRINTIN 2 MAALI	*Sensoridataa alustavasti tietokannassa, sensoridatan haku aikaleimailla pääohjelmalle toiminnassa, tietokannan testaus*												
				ma	ke	to	pe	ma	ke	to	pe		
Sprintin aikataulu	28.1.2013 - 8.2.2013		**Kalenteripäivä**	**28.1**	**30.1**	**31.1**	**1.2**	**4.2**	**6.2**	**7.2**	**8.2**		
			Sprintin päivä	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**YHTEENSÄ**	
			Päiväkohtainen resurssi tunneissa	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**56,0**	
			Päivän aikana tehty tuntimäärä	**7,0**	**7,0**	**7,0**	**7,5**	**7,5**	**7,0**	**8,0**	**8,5**	**59,5**	
Ominaisuuden/ Tehtävän ID	**Ominaisuuden/tehtävän kuvaus**	**Story Points**	**Tekijä(t)**	**Tehdyt tunnit sprintin aikana**								**Tehtävään käytetty tuntimäärä**	
10	**Datojen ottaminen tiedostosta tietokantaan**												
10.1	Tekstifilestä dataa		Miika									0,0	tämä siirrettiin tänne Sprintistä 1
												0,0	-> ominaisuutta ei tehdä opparin aikana eli se poistetaan!
12	**Graafin piirtäminen tietokannasta haetulla datalla**											**0,0**	
12.1	Eri mittausparametrien valinnat radiobuttoneista		Miika		6,0							6,0	tämä siirrettiin tänne Sprintistä 1
12.2	Monen sensorin datojen hakeminen yhteen graafiin dataseteillä					3,0	3,0	2,5				8,5	
15	**Datan hakeminen ja käyttäminen**											**14,5**	
15.1	Haetaan datat tietokannasta säikeen avulla ja välitetään eteenpäin delegaattifunktioin		Miika	4,0					1,0	1,0		6,0	
15.2	SensorDecoder luokan datajuttuja									3,0		3,0	
15.3	SQL-lause uusimman sensoridatan hakemiselle tietylle sensorille ja tietylle tyypille datetimen mukaan										5,0	5,0	
15.4	Sensoritietojen haku			1,0					0,5			1,5	
16	**UI toteutus ja julkisen tilan näyttö**											**15,5**	
16.1	Sensorin lisäys -lomake		Miika			2,0						2,0	
16.2	Datan simulointi julkisen tilan näytön graafia varten							0,5	3,5		3,0	7,0	
16.3	Julkisen tilan näytön funktiot					1,5	3,0					4,5	
16.4	Julkisen tilan näytön UI							4,5		4,0	0,5	9,0	
	Muut asiat											**22,5**	
	Publish deployment eli julkaisuversion kääntäminen Visual Studiolla		Miika	2,0	1,0							3,0	
	Softakatselmointi (tulosten näyttäminen esimiehille)						1,5					1,5	
	Koodin siivoaminen								1,5			1,5	
	Scrummin kirjoitus					0,5			0,5			1,0	
												7	

Resurssien_käytettävyys | Julkaisusuunnitelma | Kaikki tuotteen ominaisuudet | Tuotteen ominaisuuslista | Sprintin 0 tehtävälista | Sprintin 1 tehtävälista | **Sprintin 2 tehtävälista** | Sprintin 3 tehtävälista

YRITYKSEN NIMI	Oy PremiSense Ltd																
PROJEKTIN NIMI	Opinnäytetyö: tietokannan määrittely, suunnittelu ja toteutusta sensoridatalle																
SCRUM MASTER	Jussi Vepsäläinen																
PROJEKTIN AIKATAULU	10.12.2012 – 22.2.2013 + kirjoituosa 25.2.2013 - 29.3.2013																
DOKUMENTIN NIMI	scrum_seuranta_miika_kontio_UUSI.xls																
TAULUKON NIMI	Sprintin 3 tehtävälista																
SPRINTIN 3 MAALI	*Valmis hyvin pääohjelman kanssa toimiva tietokanta, toimivat käyrien piirrot graafeissa*											Kirjoittaminen alkoi pe 22.2. Ympäröity alue tarkoittaa jatkotöitä sprintin päätyttyä					
				ma	ke	to	pe	ma	ke	to	pe	ma	ke	to	pe		
Sprintin aikataulu	11.2.2013 – 22.2.2013		**Kalenteripäivä**	**11.2**	**13.2**	**14.2**	**15.2**	**18.2**	**20.2**	**21.2**	**22.2**	**25.2**	**27.2**	**28.2**	**1.3**		
			Sprintin päivä	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**YHTEENSÄ**	
			Päiväkohtainen resurssi tunneissa	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**70,0**	
			Päivän aikana tehty tuntimäärä	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**9,0**	**7,0**	**0,5**	**4,0**	**0,0**	**2,0**	**7,0**	**55,5**	
Ominaisuuden/ Tehtävän ID	**Ominaisuuden/tehtävän kuvaus**	**Story Points**	**Tekijä(t)**	**Tehdyt tunnit sprintin aikana**												**Tehtävään käytetty tuntimäärä**	
17	**Graafin fixaus**																
17.1	Graafissa olevan datan kunnollinen esittäminen y,x -akselilla		Miika	3,5			4,0	2,0						2,0	7,0	18,5	
17.2	Datan siirtely DataSeteissä. Fixataan geneerisempiin funktioihin			3,5			3,0	2,0								8,5	
17.3	Graafin x-akselin aikaleimojen korjaus				7,0	7,0										14,0	
18	**Julkisen tilan näyttö (public demo) ja siihen liittyvät asiat**															**41,0**	
18.1	Julkisen tilan näytön asentaminen ja näyttäminen asiansaisille (työkeikka)		Miika							3,0						3,0	
18.2	Julkisen tilan näytön bugien korjausta									4,0		4,0				8,0	
																0,0	
19	**Visual studiosta deployment-paketin tekeminen sekä 32bit että 64bit järjestelmiin**															**11,0**	
19.1	Publish version tekeminen 64bit ja 32bit järjestelmille. Vaatii SQLitelle sekä 32- että 64-bit -DLL-tiedostot		Miika					3,0	9,0							12,0	
																0,0	
Muut asiat																**12,0**	
	Scrummin päivitys		Miika								0,5					0,5	
																0,0	
																0,5	

Julkaisusuunnitelma | Kaikki tuotteen ominaisuudet | Tuotteen ominaisuuslista | Sprintin 0 tehtävälista | Sprintin 1 tehtävälista | Sprintin 2 tehtävälista | **Sprintin 3 tehtävälista**