Summer Camp Tietokannat

Harjoitustyö – Tietokanta ja Olio-ohjelmointi

Lappeenrannan teknillinen yliopisto
Innovation and Software (IS), LUT LBM

CT60A4301 Tietokannat & CT60A2410 Olio-ohjelmointi
Kesä 2016

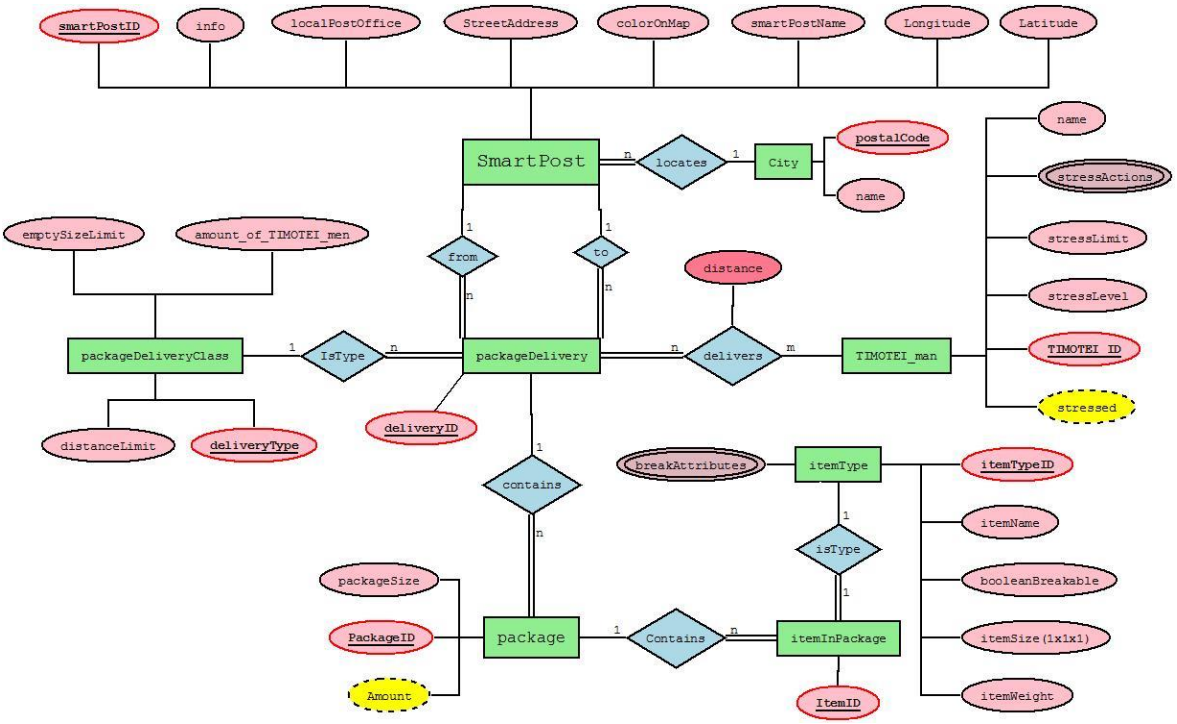
0418637 Roope Luukkainen
roope.luukkainen@student.lut.fi

SISÄLLYSLUETTELO

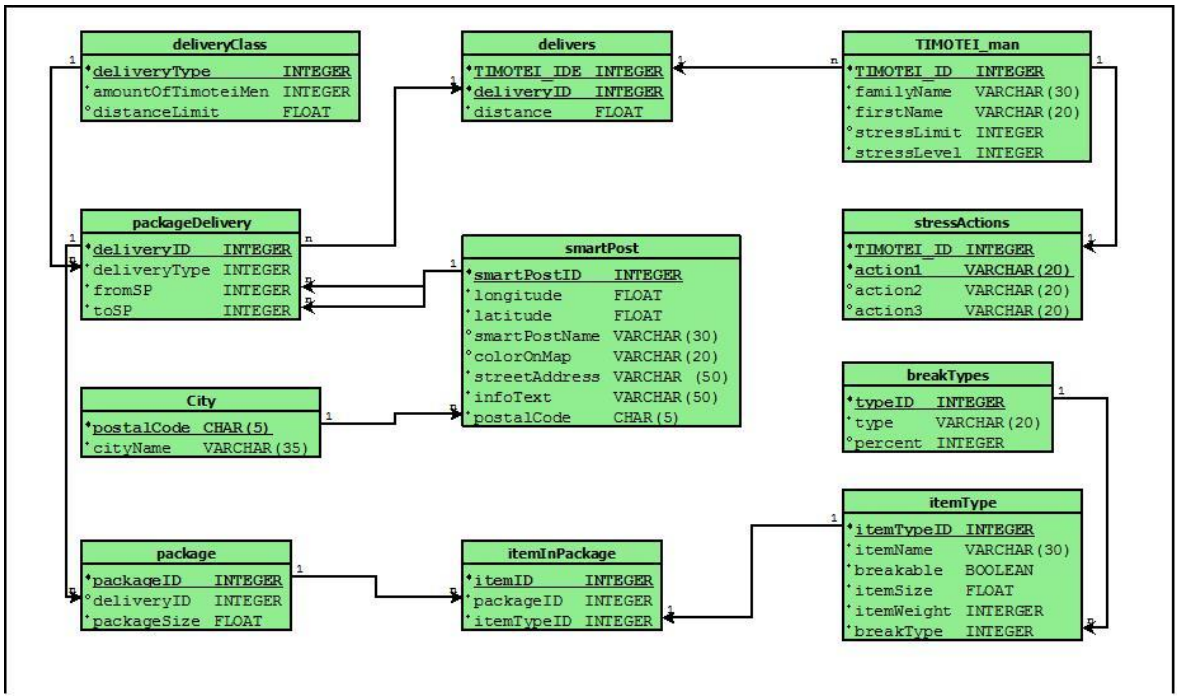
1 TYÖN KUVAUS JA RAJAUS

Tavoitteena oli luoda Toiminnaltaan Itellaa Muistuttava Ohjelmisto, Tietokantaa Edellyttävä Integraatio (TIMOTEI). Tietokantaosuus tehtiin SQL-relaatiotietokantana ja sitä käytetään Javalla ohjelmoidulla graafisella käyttöliittymällä. Pohjana koko ohjelmalle käytettiin SmartPost-verkostoa, joita piirrettiin JavaScript ohjelmointirajapintaa käyttäen Google Mapsiin JavaFX:n webView komponenttiin. SmartPost-automaattien välillä pystyy lähettämään paketteja, joiden sisällä on käyttäjän määrittämiä esineitä. Pakettikuljetuksia on kolmea eri tyyppiä ja niitä kuljettavat TIMOTEI-miehet. Tietokannan tietoja voi muuttaa ohjelman käytön aikana ja on mahdollista luoda ja poistaa SmartPosteja, paketteja ja esineitä. Kuljetuksen aikana esine voi rikkoutua ja siihen vaikuttavat monet tekijät, kuten esineen tyyppi, kuljetusluokka ja TIMOTEI-miehen stressitaso.

2 TIETOKANNAN KÄSITEMALLI JA EHEYSSÄÄNNÖT



Kuva 1 Käsitekaavio relaatiotietokannasta



Kuva 2 Taulukaavio tietokannasta

Tietokantaa toteuttaessa otettiin huomioon pää- ja vierasavainten välisten eheyssääntöjen lisäksi seuraavat eheyssäännöt:

(1) breakType voi olla vain joko normal, breaker, protective tai special.
(2) Esineen koko ja paino ovat reaalimaailmaa vastaavasti suurempia kuin 0.
(3) packageDeliverin poisto aiheuttaa myös siinä olevan paketin poiston ja paketin poisto aiheuttaa sen sisällä olevan esineen poiston.
(4) Kuljetuksen lähtö- ja kohdeautomaatti eivät voi olla yksi sama automaatti.
(5) Kuljetusta vie ainakin yksi TIMOTEI mies.

3 OHJELMAN KUVAUS

Ohjelmassa on pääikkuna, joka sisältää WebView-komponentin lisäksi lokin, mahdollisuuden valita lähetettävä paketti ja sitä kuljettava TIMOTEI-mies ja SmartPostien kartalle lisäämismahdollisuuden. SmartPostin tai sitä vastaavan kohteen voi myös luoda itse ja sille, kuten pakettien kuljetusreiteille on mahdollista valita piirtoväri. Aivan alussa ohjelma kysyy nollataanko tietokanta, eli poistetaanko kaikki SmartPostit, paketit ja esineet sekä niihin liittyvät sidokset.

Paketteja voi luoda toisessa, pääikkunasta painikkeella avattavassa ikkunassa, packageCreatorissa, jossa voi lisäksi luoda pakettiin lisättäviä esineitä. Kuljetuksen aikana esineet saattavat särkyä, mikäli ne on luotu särkyviksi valinta nappulan avulla. Esineitä voi olla neljää erilaista tyyppiä: normaali, särkevä, suojaava ja erikoinen. Esineen tyyppi vaikuttaa sekä esineeseen itseensä että muihin paketissa oleviin esineisiin seuraavasti:

(1) Särkevä esine saattaa rikkoa, muita paketissa olevia esineitä.
(2) Suojaava esine voi estää särkyvää esinettä rikkoutumasta matkan aikana.
(3) Erikoisesine särkyy helpommin, mikäli se ei ole ainoa esine paketissa.

Normaali esine ei vaikuta muihin, mutta sen voi kaikkien muidenkin tavoin luoda särkyväksi. Esineen tyypin arvoa voi lisäksi säätää luodessa siten että suojaava esine voi olla esimerkiksi 10 % tai 90 % suojaava.

4 OHJELMALLINEN TOTEUTUS JA TOIMINNALLISUUDET

Ohjelma on toteutettu Javalla, olio-ohjelmointi kielellä. SmartPosteille, esineille, paketeille, TIMOTEI-miehille ja pakettiluokille on ohjelmassa tehtävänannon mukaisesti omat luokkansa, joista tehdään tarvittavia olioita ohjelman käytön aikana. Ohjelman pääluokka on pääkäyttöliittymä ja toiseksi tärkein luokka on pakettien luonti-ikkuna. Ohjelmassa käytetään Index.html tiedostoa, joka on JavaScript tiedosto Google Maps rajapintana. Tein pieniä muutoksia tiedostoon kuten lisäsin Markereiden poisto function, Markerin piirto toteutetaan tietokannasta luettaessa koordinaattien, ei osoitteen perusteella. Suurin osa tietojen tallentamisesta ja hakemisesta tehdään DBHandler-luokan (tietokanta käsittelijän) kautta. Tietokantaan tehtävät kyselyt ja toiminnot ovat:

(1) Lisää SmartPost-kohde tietokantaan:

a. INSERT INTO smartPost (smartPostName', 'info', 'streetAddress', 'postalCode', 'latitude', 'longitude') VALUES (?, ?, ?, ?, ?, ?);

(2) Lisää kaupunki tietokantaan:

a. INSERT INTO city ('postalCode', 'cityName') VALUES (?, ?);

(3) Lisää esinetyyppi tietokantaan ja esine pakettiin.

a. INSERT INTO itemType ('itemName', 'itemSize', 'itemWeight', 'breakable', 'breakType' VALUES (?, ?, ?, ?, ?);

b. INSERT INTO itemInPackage ('itemTypeID', 'packageID' VALUES (?, ?);

(4) Lisää paketti ja paketin kuljetus tietokantaan.

a. INSERT INTO package ('deliveryID', 'packageSize') VALUES (?, ?);

b. INSERT INTO packageDelivery ('deliveryType', 'fromSp', 'toSP') VALUES (?, ?, ?);

(5) Tietokantaan tulee lisätä TIMOTEI-miehet, stressitoiminnot, lähetysluokat ja esineiden särkymistyypit. Näistä viimeistä lukuun ottamatta kaikki syötetään kokonaisuudessaan tietokantaan jo sitä luodessa.

a. INSERT INTO "TIMOTEI_man" ("familyName", "firstName", "stressLimit", "stressLevel") VALUES (?, ?, ?, ?);

b. INSERT INTO "stressActions" ("TIMOTEI_ID", "action1", "action2", "action3") VALUES (?, ?, ?, ?);

c. INSERT INTO deliveryClass (‘deliveryType’, ‘amountOfTimoteiMen’, ‘emptySizeLimit’) VALUES (?, ?, ?);
d. INSERT INTO breakTypes ('btype', 'percent') VALUES (?, ?);

(6) Kaikkien lisättyjen tietojen lukeminen tietokannasta.

a. SELECT [attribuutit] FROM table [lisämääreet, esim. WHERE lause]

(7) Kohteiden muokkaaminen ja poistaminen tietokannasta.

a. UPDATE table SET attribuutti = ? WHERE lisämääre = ?;
   Ensimmäinen ? on uusi arvo ja toinen kertoo WHERE-ehdon.
b. DELETE FROM table WHERE lisämääre = ?;

Lisämääre on attribuutin nimi, jota WHERE ehdossa verrataan.

Kaikki kysymysmerkit ovat arvojen välittämistä tietokantaan parametreina ja nämä arvot välitetään DBHandlerille metodikutsussa. Yhteys tietokantaan luodaan Javan Connection muuttujalla ja tietokantakyselyt rakennetaan Javan PreparedStatement muuttujalla, jonka tuloksena saadaan ResultSet, josta voidaan Javassa lukea tulokset.

Tietokantaan luodut näkymät (view) luodaan yksinkertaistamaan SELECT lauseita tietokannasta lukemisessa. Näkymät luodaan seuravilla komennoilla:

(1) CREATE VIEW "smartPostView" AS
SELECT smartPostID, longitude, latitude, smartPostName, colorOnMap, sp.postalCode, cityName, streetAddress, info
FROM smartPost sp JOIN city c ON sp.postalCode = c.postalCode;

(2) CREATE VIEW "packageView" AS
SELECT packageID, pd.deliveryID, packageSize, deliveryType, s.smartPostID[from_ID], s.smartPostName[from_smartPost], s.streetAddress[from_Address], s.postalCode[from_ZIP], s.cityName[from_City], sp.smartPostID[to_ID], sp.smartPostName[to_smartPost], sp.streetAddress[to_Address], sp.postalCode[to_ZIP], sp.cityName[to_City]
FROM package p
JOIN packageDelivery pd ON p.deliveryID = pd.deliveryID
JOIN smartPostView s ON s.smartPostID = pd.fromSP

```
JOIN smartPostView sp ON sp.smartPostID = pd.toSP;
```

(3)
```
CREATE VIEW "itemTypeView" AS
SELECT itemTypeID, itemName, breakable, itemSize, itemWeight, btype, percent
FROM itemType i
JOIN breakTypes b ON i.breakType = b.breakTypeID;
```

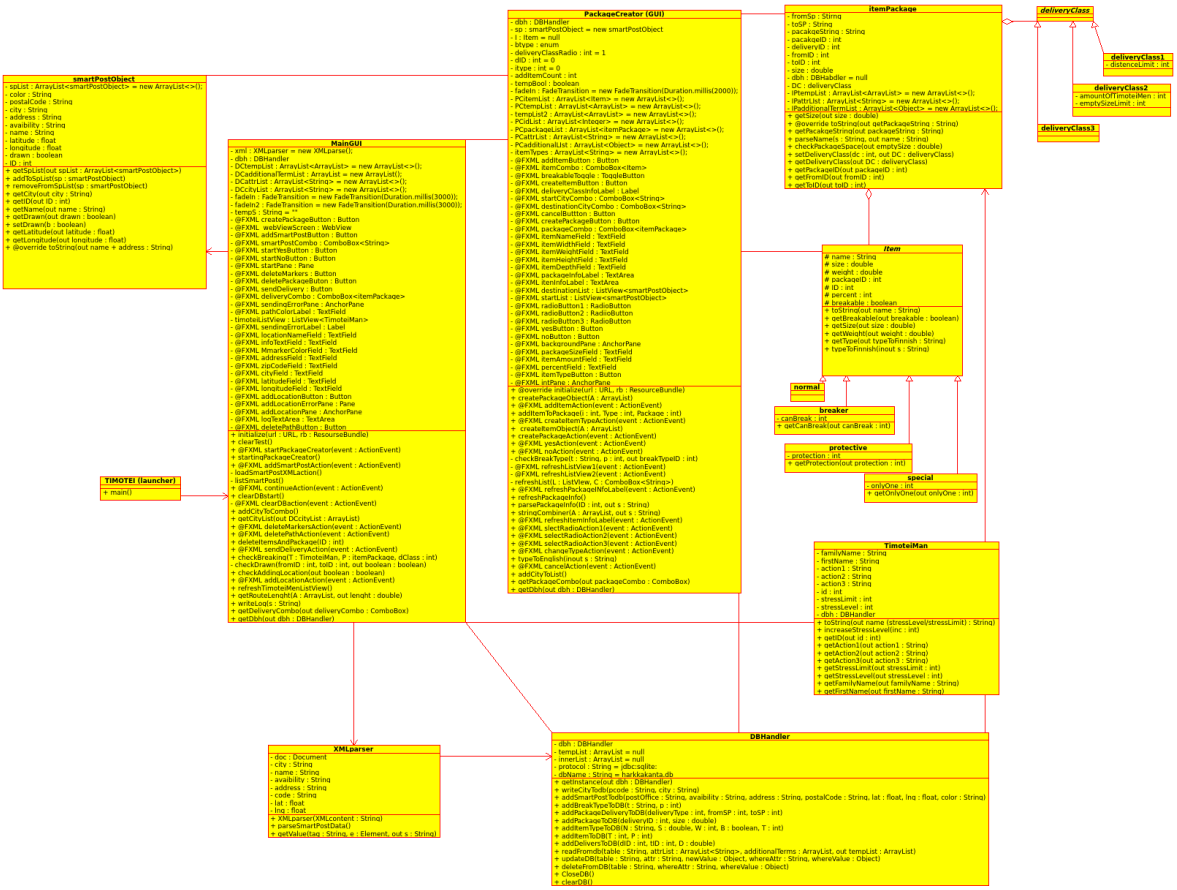
(4)
```
CREATE VIEW "inPackagesView" AS
SELECT ip.packageID, itemID, itemName, btype[breakType], percent, breakable,
itemSize, itemWeight, from_ID, from_smartPost, to_ID, to_smartPost,
deliveryType
FROM itemInPackage ip
JOIN itemTypeView it ON ip.itemTypeID = it.itemTypeID
JOIN packageView p ON ip.packageID = p.packageID
ORDER BY ip.packageID, itemName;
```

(5)
```
CREATE VIEW "timoteiManView" AS
SELECT T.TIMOTEI_ID, firstName, familyName, stressLimit, stressLevel,
action1, action2, action3
FROM TIMOTEI_man T
JOIN stressActions s ON T.TIMOTEI_ID = s.TIMOTEI_ID;
```

Ohjelmassa olevat toiminnallisuudet ovat seuraavat:

(1) SmartPostien piirtäminen kartalle.
(2) Oman SmartPostin tai sitä vastaavan kohteen luominen.
(3) Paketin luominen valiten, sille koko ja lähtö- ja kohdeautomaatit.
(4) Esinetyypin luominen ja tyyppien mukaisten esineiden lisääminen pakettiin.
(5) Paketin poistaminen.
(6) Paketin lähettäminen määritettyjen automaattien välillä ja sitä kuljettamaan valitaan TIMOTEI mies.
(7) Tietokannan nollaaminen joka poistaa SmartPostit, esineet ja paketit tietokannasta.

5 LUOKKAKAAVIO



Kuva 3 Ohjelman täydellinen luokkakaavio kaikkine metodeineen ja attribuutteineen.

6 YHTEENVETO

Ohjelman toteutus SQL relaatiotietokannan yhdistäminen Javan olio-ohjelmointiin oli kokonaisuudessaan varsin mielenkiintoista. Ongelmia kohdattiin tietokanta toteutuksessa esineiden lisäämisessä pakettiin, sillä alun perin kukin esine oli lopullista esinetyyppiä vastaava, jolloin esineen lisääminen pakettiin vaati aina uuden esineen luomisen. Uudella tavalla esinetyyppiä vastaavia esineitä voidaan lisätä useita samaan tai eri paketteihin. Toinen merkittävä muutos alkuperäiseen ER-malliin oli erottaa kaupungit omaan tauluunsa normalisoinnin lisäämiseksi ja SmartPostilla on vain postinumero, joka määrää sitten kaupungin. Tässä on kuitenkin teoreettinen mahdollisuus saada aikaan virheitä, sillä kuten postin sivulla: http://www.posti.fi/henkiloasiakkaat/asiakaspalvelu/postin-jakelu-ja-postinumerot.html lukee: *”Postinumeroalueenrajat eivät ole samat kuin kuntarajat, sillä sama postinumero voi jakaantua esimerkiksi kahden tai useamman kunnan alueelle. Samoin kunnan alueella voi olla useita erilaisia osoitetoimipaikan nimiä.”*
Kuitenkin tarkistettuani kaikki nykyiset SmartPost-automaatit ei mikään niistä osunut alueelle, jossa samalla postinumerolla voisi olla useita eri postitoimipaikkoja.
SmartPosteilla oli aluksi tarkoitus olla pääavaimena leveys- ja pituuspiirin yhdistelmä eikä suoranaista ID:tä, koska kukaan tuskin saa samaan koordinaattiin tungettua kahta SmartPost-automaattia ellei niitä useampaan kerrokseen laiteta. ID kuitenkin helpotti vierasavain viittauksia, joten toteutin SmartPostien identifioinnin sillä tavoin.

Haastavin osuus koko toteutuksessa oli suunnitella oliotyyppinen ohjelma, sillä kyseessä oli ensimmäinen hitusen suurempi olio-ohjelma. Lopputuloskin olisi voinut olla olio-ohjelmointi osion osalta olla hyvin paljon parempi sekä suunnittelun että ohjelmoinnin kannalta. Näin jälkeenpäin olisi tehnyt erikseen molemille GUI:lle pelkkiin JavaFX komponentteihin liittyviä metodeja ja sitten erikseen luokat varsinaisille toteutuksille.
Virheenkäsittelijä ja tarkastusluokka olisivat selkeyttäneet ohjelmaa ja lisäksi tekisin yhden virhe pop-up –ikkunan, jonka tekstiä muuttaisin virheestä riippuen enkä useita sekavia Labeleita tulemaan näkyviin sinne tänne ohjelmaa käyttäjävirheen sattuessa. Olisin lisännyt enemmän tarkistuksia käyttäjäsyötteisiin, mutta olen tyytyväinen CHECK-lauseisiini tietokantaosuudessa.

Muita havaittuja asioita: Alun perin en lainkaan luonut esine, paketti tai kuljetusluokkia Javaan, vaan olisin suorittanut kaiken tarvittavan tiedon haun tietokannasta, mutta tehtävänanto vaati näiden luokkien toteuttamista, joten näin tein. En täysin ymmärrä, miksi pitäisi luoda tietokantaan tiedot, joista käytännössä kaikki tiedot luodaan myös itse käyttöliittymän ohjelmassa jonkinlaiseen listaan. Eikö tällöin tiedot ole turhaan kaksi kertaa tallennettuna.

SQLite3 ei näytä tukevan Arrayta vaikka SQL-kieli sitä ymmärtääkseni tukeekin. Olisiko järkevää tallentaa esim. esineen koko [x, y, z] Arrayna yhteen sarakkeeseen, koska en nähnyt tarpeelliseksi kolmea eri saraketta varata koon tallentamiseen, sillä esine on yhtä suuri eri asennoissakin, jolloin arvot kuitenkin muuttuisivat.

SQLite3 ei myöskään näyttänyt välittävän numeroiden tyypeistä vaan kaikki tallennettiin REALina tietokantaan kuten myös CHAR, VARCHAR ja kaikki muut tekstityypit tallennettiin yksinkertaisesti TEXTiksi, jolloin pituusrajoitus/määritys ei millään tavalla toiminutkaan halutusti.

Totuusarvot näyttivät myöskin olevan suoraan 1 ja 0 muodossa eikä true/TRUE tai false/FALSE muodossa vaikka näissä kaikissa muodoissa tiedot voitiin tietokantaan syöttää.

Hyvän olio-ohjelmointi tyylin noudattaminen olisi huomattavasti helpompaa, mikäli olisi hyvä malliesimerkki siitä, mitä/mikä on oikeasti hyvä olio-ohjelmointityyli käytännössä. Yksittäiset kurssivideoissa esiin nousseet mallit olivat tietysti erinomaisia. Hieman suuremman, hyvin kommentoidun ja toteutetun ohjelman esittely auttaisi varmasti ymmärtämään vielä paremmin olioparadigmaa. Internet on toki pullollaan olio-ohjelmia, mutta harvat niistä ovat ”todistetusti” hyvin toteutettuja ja näin koko päiväisen opiskelun ohella ei kaikkea vapaa-aikaa enää kyennyt kuluttamaan tällaisten asioiden etsimiseen.

Olio-osuus koko työstä tuntui huomattavasti suuremmalta kuin tietokantaosuus. Tietokanta osuuden (1 & 2 osuus työstä) toteutin yhden päivän aikana ja sen muuttaminen ja Javan PreparedStatentteihin syötetyt tietokanta kutsut olivat vain muutaman tunnin homma kokonaisuudessaankin. Itse Java ohjelmointiin sen sijaan kului jopa viikkoja aikaa. Yritin ehkä hieman turhan laajaa ohjelmaa toteuttaa ja olen melko hidas ohjelmoimaankin, mutta tehtävänanto oli sen verran pirstaleinen että aluksi oli vaikea hahmottaa ja listata mitään vaadittavia yksityiskohtia ohjelmasta.