Olio-ohjelmoinnin ja tietokantojen harjoitustyö

Roman Hyvönen

Lappeenrannan teknillinen yliopisto

Tietotekniikan koulutusohjelma

Kevät, 2016

Sisällysluettelo

Tehtävänannon kuvaus ja työn rajoitukset

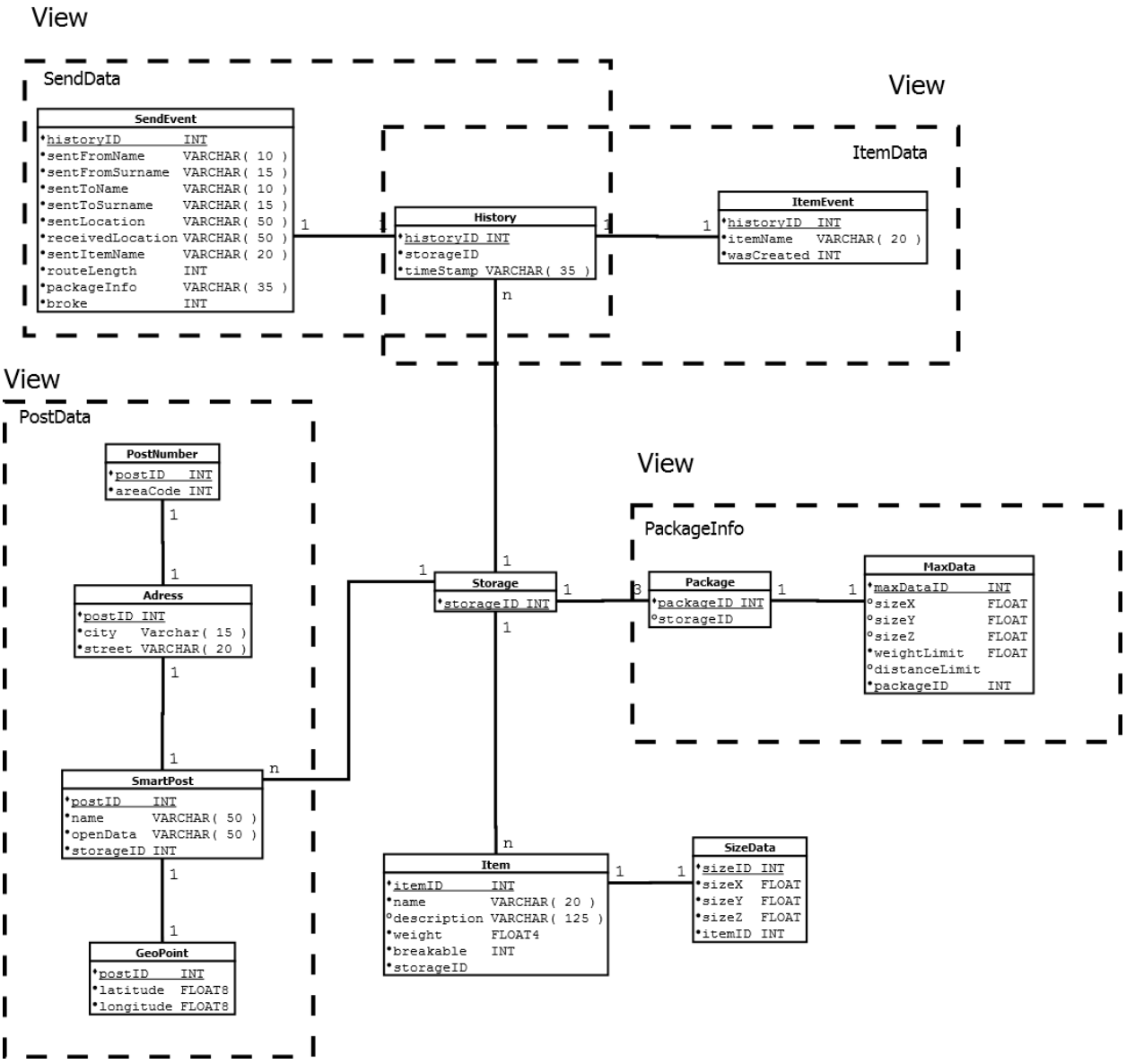
Harjoitustyön tehtävänä on toteuttaa tietokantoja hyödyntävä SmartPost-simulaatio kuvitteelliselle loppuasiakkaalle. Tehtävässä esiintyi useita eri vaatimustasoja, jotka riippuivat halutusta pistemäärästä. Laajimpaan niistä kuuluivat seuraavat ominaisuudet:

- SmartPostit ladataan internetistä tietokantaan.
- Käyttäjä kykenee lisäämään ja poistamaan yksittäisiä automaatteja (kartalla).
- Esineitä/paketteja pystytään luomaan/poistamaan.
- Esineitä voidaan lähettää eri luokan paketeissa, joilla on omat rajoituksensa.
- Lähetykset voivat hajota.
- Käyttäjän toiminnoista pidetään lokia.
- Ohjelman mennessä kiinni kirjoitetaan kuitti.
- Ohjelman alkutilassa käyttäjä voi käyttää vanhaa tietokantaa (ja lokia) tai ladata uuden.
- Ohjelman ulkoasulla on vakiota mutkikkaampi tyyli (CSS)

Tietokannan käsitemalli ja eheyssäännöt

Käsitemalli

Tietokannan on pidettävä sisällään tietoa jokaisesta jokaisesta SmartPost-oliosta ja kaikista sen attribuuteista. Käyttäjä voi lisätä uusia esineitä, jotka on pystyttävä tallentamaan tietokantaan myös. Lähetyspaketin tyyppejä on kolme ja niistä jokaisella on omat rajoituksensa. Niiden lisäksi tietokannan tulee pitää kirjaa lokista, jotta se voidaan ladata kätevästi ohjelman käynnistyessä.



Kuva 1. Tietokannan käsitemalli.

Tietokantaan tehdään useita JOIN-kyselyjä, joita helpottamaan on tehty neljä kuvassa 1 esillä olevaa näkymää. SQLite ei tue näkymiin inserttaamista, joten jokainen insert tapahtuu erikseen taulukohtaisesti. Skeeman keskellä sijaitseva Storage-taulu yhdistää kaikki skeeman osiot loogisesti toisiinsa.

Eheyssäännöt

Eheyssäännöt sai toteuttaa joko ohjelmallisesti, tai tietokannassa. Lisäsin tietokantaan tyyppimääritteet (sqlite ei rajoita tyyppejä oletuksena) ja NOT NULLin valtaosaan tauluista. Rajoitukset ja tyyppien hallinnan toteutin ohjelmallisesti. Muutama ylimääräinen tarkistus oli mielestäni miellyttävämpi vaihtoehto SQL-virheiden parsimiseen nähden. Alapuolella on lista ohjelman eheyssäännöistä ja niiden ylläpitämisestä.

- Pää- ja vierasavaimet:

Avaimien eheyssääntöjen tärkein kohta on koheesio: oikea pääavain vastaa juuri oikeaa vierasavainta. Tästä eheydestä pidin huolta kahdella eri tavalla. Ensimmäinen oli autoincrementin ja rollbackin käyttö 1-1 tauluissa (SamrtPostien tiedot). Jos transaktio onnistuu, niin jokainen taulu saa autoincrementin avulla oikean pääavainarvon. Mikäli yksikin transaktion osa epäonnistuu, jätetään koko lisääminen tekemättä. Esine- ja lokitauluissa päätin pitää avaimista huolen ohjelmallisesti kokeilun vuoksi. Ladatessaan tietokantaa ohjelma lukee jokaisen avaimen ja asettaa korkeimman muistiin. Kun tietokantaan lisätään uusi esine, tai loki ohjelma antaa seuraavan vapaan avaimen pääavaimeksi.

- Tietotyypit:

Ohjelman toiminnollisuuden kannalta on tärkeää, että tietotyypit ovat oikeita. Postinumerot voivat alkaa nollalla, joten ne eivät voi olla int-tyyppisiä. Suoraan ohjelmasta tulevan datan tietotyyppien hallinta on triviaalia. Ainoa erityistapaus oli totuusarvojen – joita SQLite ei tue - muuttaminen kokonaisluvuiksi. Käyttäjän syötteissä voi tulla vääriä tietotyyppejä, jotka sotkisivat ohjelman toiminnan. Niitä valvoo GUI lukitsemalla kaikki napit, jotka lähettävät dataa tietokannalle, ennen kuin jokainen syöte on oikeaa tyyppiä ja miellyttävältä arvoväliltä, jos sellainen on määritelty.

- Taulujen väliset suhteet:

Jokaista SmartPost-taulua kohden on oltava tasan yksi GeoPoint, Adress sekä PostNumber.

Jokaista Item-taulua kohden on oltava yksi SizeData.

Jokaista Package-taulua kohden on oltava yksi maxSizeData.

Jokaista History-taulua kohden on oltava yksi ItemEvent, tai SendEvent, muttei molempia.

Jokainen SmartPost-, Item-, Package-, ja History-taulu ydistyy yhteen ja samaan Storage-tauluun. Jonka pääavain on 1.

Ohjelman kuvaus

Ohjelma alkaa antamalla käyttäjälle pop-up ikkunan, jossa tämä valitsee aloitetaanko vanhasta tilasta, vai lähdetäänkö liikkeelle puhtaalta pöydältä. Käyttäjän valinnasta riippuen luodaan joko uusi tietokanta ja asetetaan sinne vakio esineet ja paketit, tai otetaan vanha käyttöön. Tämän jälkeen esille tulee ohjelman pääikkuna. Sitä kautta käyttäjä näkee koko ajan joko kartan tai lokin, sekä ohjelman hallinta-asetuksia ikkunan oikealla puolella. Käyttäjä kykenee lisäämään postitoimipaikkojen markkereita kartalla ja näkemään niiden tiedot niin kartalta, kuin myös valinta paneelista. Käytössä olevat postitoimipisteet voidaan myös poistaa.

Painamalla ”Manage Items” käyttäjä pääsee luomaan ja poistamaan uusia esineitä, joille voi määrittää erinäisiä attribuutteja. ”Send Package” vie käyttäjän lähetysruutuun, jossa määritetään lähettäjän, vastaanottajan ja itse lähetyksen tiedot. Jokainen toiminto kirjataan ylös lokiin ohjelman ajon aikana ja lähetetään tietokantaan seuraavaa käynnistystä varten. Käytön jälkeen ohjelma tulostaa kyseisen istunnon lokin ulkopuoliseen tekstitiedostoon käyttäjän myöhempää käyttöä varten.

Raportti ohjelmallisesta toteutuksesta ja lista toiminnallisuuksista

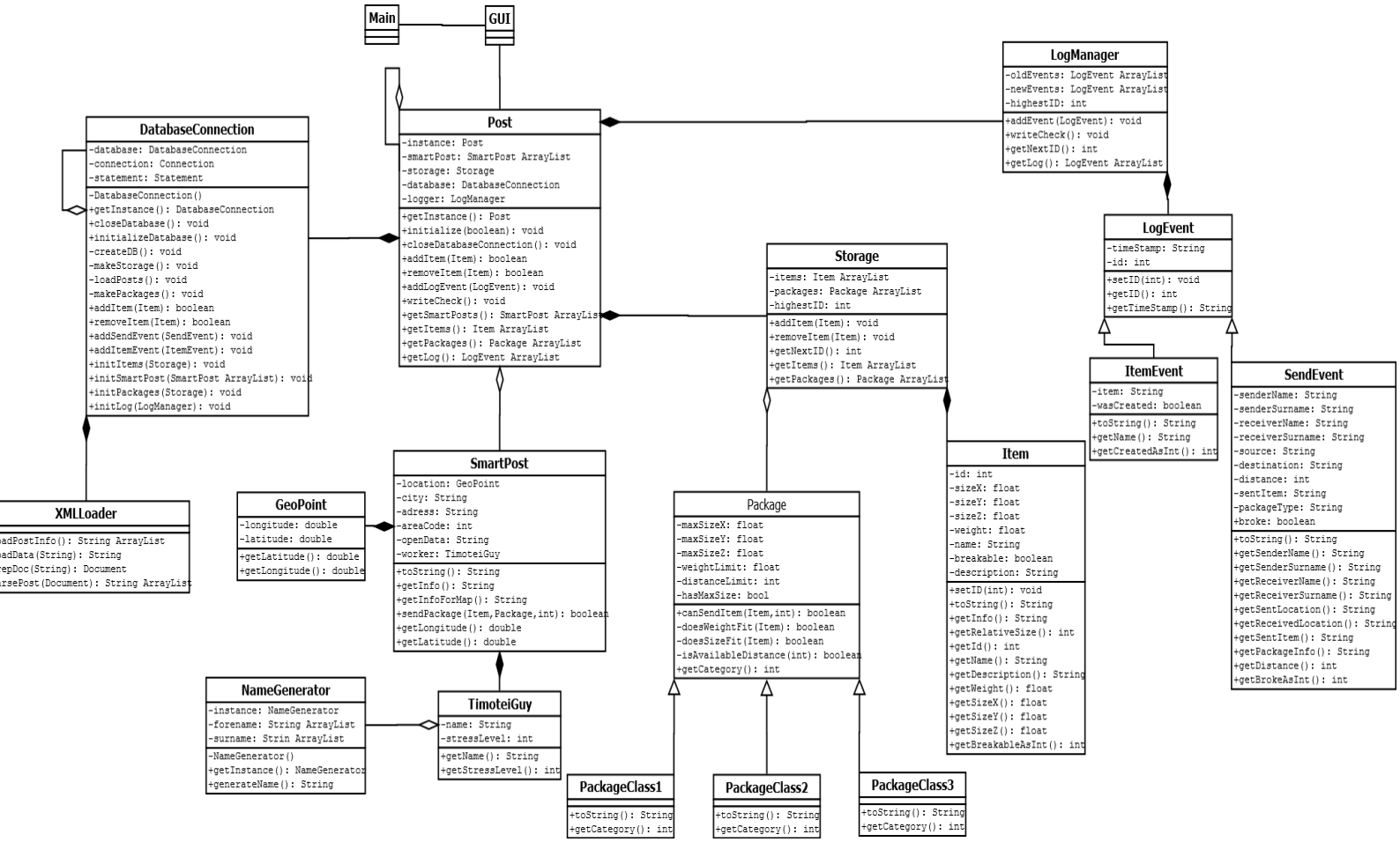
Aloitin ohjelmallisen toteutuksen miettimällä miten saisin aikaan mahdollisimman helposti hallittavan oliohierarkian. Päädyin käyttämään ”Post”-luokkaa rajapintana GUI:n ja loogisen toteutuksen välillä. GUI antaa käskyjä postille käyttäjän toimien mukaan ja posti ajaa niitä eteenpäin. Näin GUI:n ei tarvitse tietää kuin yhdestä oliosta ja sen käyttäminen on yksinkertaista.

Post-luokka pitää sisällään tiedon kaikista yksittäisistä SamrtPosteista, varastosta, lokista ja tietokannan käsittelijästä. Kaikki nämä luokat ovat yksityisiä, eikä niitä lähetätä postin ulkopuolelle, poikkeuksena SmartPost arrayList ja tietokannasta ladatun lokin tiedot comboBoxien alustamista varten. Mitään ei kuitenkaan muokata Post-luokan ”yläpuolella”. Tietokannasta lataaminen tapahtuu referenssien avulla: Post kutsuu alaluokansa DatabaseConnection initialize-funktioita ja lähettää referenssinä arrayListit, tai lokin/tavaroiden hallitsijaluokat, joiden täytyy lisätä lopulliset tiedot itse tietokannan eheyssääntöjen hallinnan vuoksi.

Toiminnallisuudet

- XML-tiedon lataaminen verkosta
- Tietokannan alustaminen
- Tietokantaan syöttäminen ja sieltä lukeminen
- Luokkien ”lataaminen” tietokannasta
- Esineiden lisääminen ja poistaminen
- SmartPostien aktivoiminen/deaktivoiminen
- TimoteiMiesten nimigeneraattori ja stressitason käyttäminen
- Pakettejen lähettäminen/hajoaminen
- Piirrettyjen reittien poistaminen
- Lokin pitäminen käyttäjän toimista
- Kuitin kirjoittaminen

Ohjelman täydellinen luokkakaavio



Kuva 2. Ohjelman UML-luokkakaavio.

Luokkakaavion tutkiminen suoraan raportista voi olla hankalaa. Ohjelman pääkansiossa on sama luokkakaavio .dia- ja.png-muodossa. GUI:lla on valtava määrä (50+) muuttujia ja funktioita, joiden esillä oleminen ei anna melkein mitään järkevää tietoa ohjelman toiminnasta ilman sceneBuilderia, tai vastaavaa ohjelmaa, josta näkee jokaiseen objektiin liittyvät nimet ja funktiot. Tein tietoisen päätöksen jättää ne pois luokkakaaviosta selkeyden lisäämiseksi ja kuvan koon järkevänä pitämiseksi.

Yhteenveto

Harjoitustyön ohjelma oli laajin ja monipuolisin ohjelma mitä olen tähän asti tehnyt. Jouduin kaivamaan paljon tietoa kurssimonisteista ja googleista, sekä käyttämään täysin uusia graafisia komponentteja ja ohjelmointitekniikoita.

Opin ohjelman huolellisen suunnittelun merkityksen. Tein laajan UML:n ja tietokantojen skeeman ennen ohjelmointia, mutta molemmat kokivat lukuisia muutoksia kun löysin parempia ja parempia tapoja toteuttaa asioita. Seuraavalla kerralla alkuperäiset suunnitelmat tulevat olemaan lähempänä lopullista ohjelmaa.

Havaitsin tietokannan vaativat hirveän määrän gettereitä, joita ei muuten tarvittaisi ja aion jatkossa miettiä tapoja välttyä ylimääräisiltä funktioilta tietokantojen kanssa. Minimalistiset, mutta täydelliset luokat ovat tavoittelemisen arvoisia.

Relaatiotietokannat nostivat esille useita suunnittelukysymyksiä, joista väiteltiin suuntaan tai toiseen Stack Overflowin syvyyksissä. Sovelsin parhaani mukaan kurssimateriaaleja ja jätin epävarmat menetelmät käyttämättä. Saman pääavaimen käyttäminen useissa tauluissa oli suurin kysymys. Monien lähteiden mukaan se on sallittu menetelmä ja päätin käyttää sitä SmartPosteille, sekä lokille. Kolmas normaalimuoto toteutuu näin, sillä kaikki ei pääavaimesta funktionaalisesti riippuvat attribuutit on hajautettu eri tauluihin, mutta lopulta ne riippuvat kuitenkin samasta postiID:stä.

Toivotut arvosanat ja perustelut

Olio-ohjelmointi:

Suoritin 40 pisteen vaatimuksista seuraavat:

- Laajennettu GUI
- Kirjanpidon lataaminen ohjelman käynnistyessä

Näiden lisäksi tein kaikki muiden pisteiden vaatimukset. 26 pisteen vaatimuksen ”oma hieno feature” on Timotei miehet, joista jokaisella on oma nimi ja stressitaso, joka vaikuttaa paketin postitukseen. Mikäli tekninen toteutukseni on riittävän hyvä toivon täysiä pisteitä.

Tietokannat:

Käsitemalli koostuu 12 yksilötyypistä, neljästä näkymästä ja useista JOIN-kyselystä. Eheyssäännöistä pidetään huolta joko ohjelmassa(pääosin), tai tietokannassa riippuen taulusta. Tietotyyppien eheyttä hallitaan tarkkailemalla käyttäjän syötteitä ja avaimien eheyttä rollbackkien käytöllä. Jos attribuuteilla on maksimiarvoja, niitä hallitaan ohjelmassa.

Käsitemalli on luettavissa tiedostosta schema.cpp (schema.sql on tehty eri tekstieditorilla, eikä se näytä tabeja oikein) ja kaikki kyselyt/insertit/deletet löytyvät DatabaseConnection.java:sta.

Toivon kiitettäen hyväksyttyä arvosanaa.