CT60A4301 Tietokannat

Harjoitustyö 1 – Tietokannan suunnittelu

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

CT60A4301 Tietokannat
Kesä 2016

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

SISÄLLYSLUETTELO

1 MÄÄRITYS

TIMOTEI-projektissa luotiin SmartPost-automaattien väliseen pakettien lähetykseen suunniteltu sovellus. Tietokanta ylläpitää tiedon SmartPostien geograafisesta sijainnista, nimestä, osoitteesta ja muusta vapaavalintaisesta informaatiosta, kuten aukioloajoista. Normalisaation kannalta kaupunki on erillisessä taulussa, jonka pääavaimena toimii postinumero. Tietokantaan voidaan lisätä uusia SmartPosteja ja omia vapaavalintaisia kohteita, jotka ovat verrattavissa SmartPosteihin. Pakettien lähetys automaattien välillä vaatii tietokannan ylläpitämään tietoa paketeista ja niiden sisällöstä eli esineistä sekä näihin liittyvistä attribuuteista kuten koosta ja painosta. Esineitä on neljää eri päätyyppiä: tavallinen, särkevä, suojaava ja erikoinen, joista jokaisella on jokin vaikutus muihin paketissa oleviin esineisiin. Paketteja kuljetetaan ja niillä on aina kuljettaja, TIMOTEI-mies, joka ovat myös kirjattuna järjestelmään. Kuljetukset ovat aina jonkin yhdestä kolmeen luokan lähetyksiä ja niillä on lähtö- ja kohdeautomaatti. Koska TIMOTEI-miehet ovat inhimillisiä olentoja voivat he joissain tilanteissa käyttäytyä aggressiivisesti paketteja kohtaan suorittamalla stressinpurku toimenpiteitä, jotka ovat yksilöllisiä. Tarkemmat tiedot tietokannan rakenteesta näkyvät käsitekaaviosta ja tietokantaa havainnollistavasta taulukaaviosta.

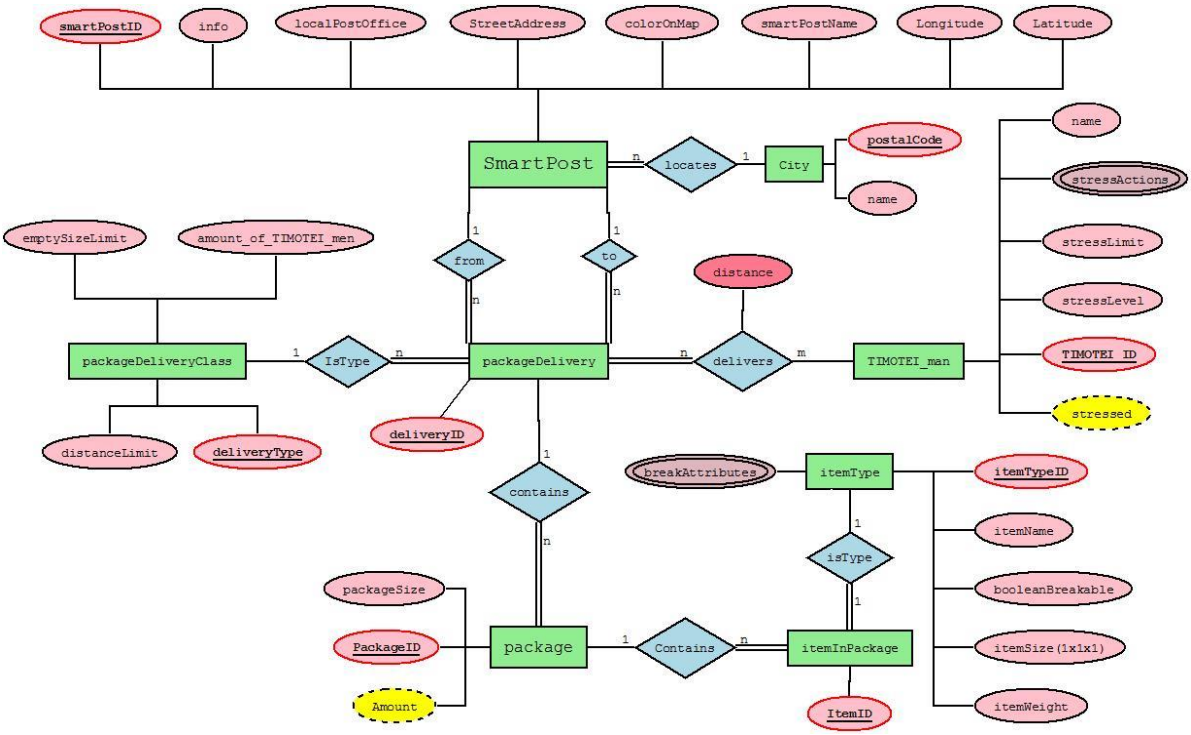
Sovelluksen pohjana oleva tietokanta kevyt tietokanta ja sitä käyttävät paketteja lähettävät ihmiset SmartPostien yhteydessä. Käytännössä käyttäjäkunta on siis hyvin laaja, mutta yhtä aikaisten käyttäjien määrä on pääosin varsin pieni. Käyttäjät eivät voi missään tilanteessa tehdä suoria kyselyitä SQL-tietokantaa, mutta heidän lisäämät, poistamat ja lähettämät paketit vaativat kyselyitä tietokantaa, joiden perusteella käyttäjälle kerrotaan tarvittavia tietoja paketeille ja/tai esineille tehtävistä toimenpiteistä.
Seuraavat toiminnot toteuttavat tietokanta komennot löytyvät:

(1) Lisää SmartPost-kohde tietokantaan.
(2) Lisää kaupunki tietokantaan.
(3) Lisää esinetyyppi tietokantaan ja esine pakettiin.
(4) Lisää paketti ja paketin kuljetus tietokantaan.
(5) Tietokantaan tulee lisätä TIMOTEI-miehet, lähetysluokat ja esineiden päätyypit.
(6) Kaikkien lisättyjen tietojen lukeminen tietokannasta.
(7) Kohteiden muokkaaminen ja poistaminen tietokannassa on onnistuttava.

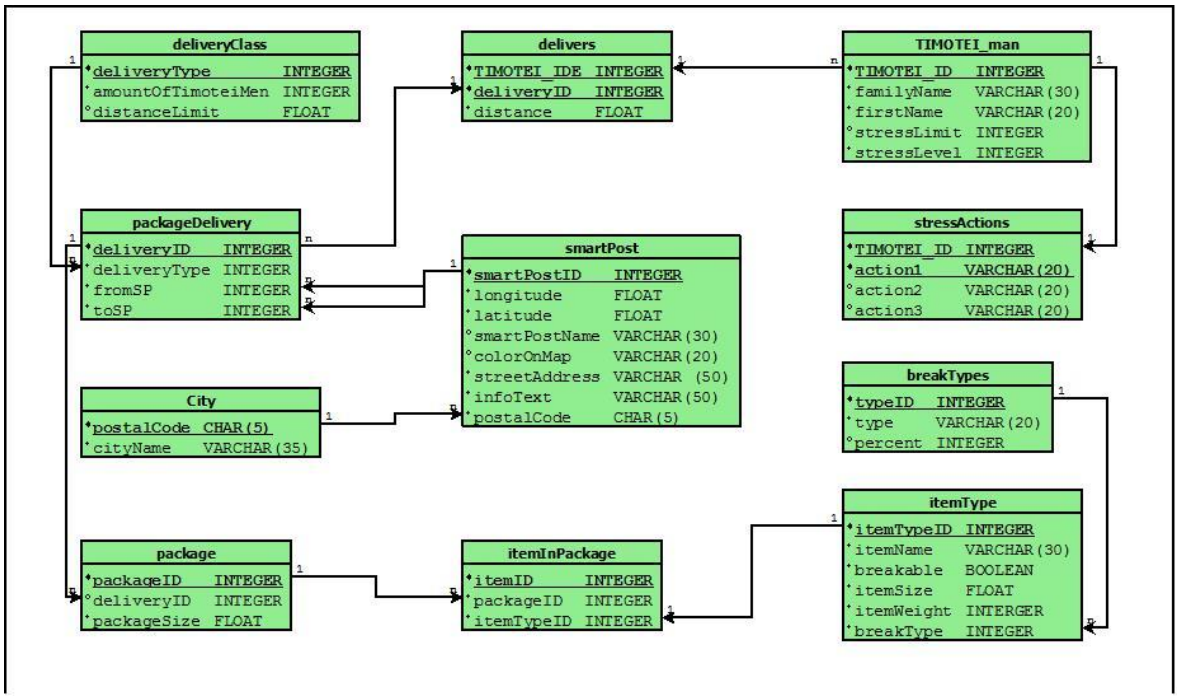
Kohta 6 on ohjelmakoodin toiston välttämiseksi toteutettuna siten että annetaan taulun nimi, attribuutit ja mahdolliset lisämääreet (kyselyn WHERE ehdot). Kohdan 7 komennot on toteutettu 6 vastaavasti siten että aluksi annetaan taulun nimi, sitten attribuutti ja lopuksi lisämääre. Tiedon päivityksessä lisäksi uusi päivityksen jälkeinen arvo.

Kyselyiden yksinkertaistamiseksi on luotu näkymiä, jotka muodostetaan jopa monimutkaisilla JOIN lauseilla, mutta näin varsinaiset kyselyt ovat hyvin yksinkertaisia.

2 KÄSITEMALLI



Kuva 1 Käsitekaavio relaatiotietokannasta



Kuva 2 Relaatiotietokantaa kuvaava taulukaavio

Alleviivattu on pääavain. Täysi ympyrä ● ei voi olla NULL ja avoin ○ voi. Nuolen pää on renkitaulun puolella ja osoittaa siis vierasavaimeen. Nuolen vieressä suhteen arvot (1, n).

Eheyssäännöt, joita en näe ylläolevista kuvista ovat seuraavat:

(1) breakType on vain jokin seuraavista: normal, breaker, protective tai special.

(2) Esineen koko ja paino ovat suurempia kuin 0.

(3) packageDeliverin poisto aiheuttaa myös siinä olevan paketin poistin ja paketin poisto aiheuttaa sen sisällä olevan esineen poiston.

(4) Kuljetuksen lähtö- ja kohdeautomaatti eivät voi olla yksi sama automaatti.

(5) Kuljetuksella on aina vähintään yksi TIMOTEI mies. (Käsitekaaviossa on n-m suhde, mutta jos näin jälkikäteen olen ymmärtänyt oikein, on n yhdestä ylöspäin ja m nollasta ylöspäin, joten suhteen tulisi olla n-n. Aluksi ymmärsin että käytetään eri kirjaimia vain erottaakseen ne toisistaan.)

3 TIETOKANTATOTEUTUS

Toteutustapa: Tietokantaa käytetään javalla tehtyllä ohjelmalla, jonka koodi on täysin itse tehtynä. Ohjelman suunnittelun pohjalla on käytetty tehtävänannossa ollutta UML-kaaviota ja case-kuvausta, mutta muutoksia on tehty esimerkiksi siinä että paketilla ei ole erikseen kolmea eri luokkaa, vaan on erikseen paketin kuljetusluokkia.

Yleistä: Relaatiomalliksi viedessä, huomasin ongelmana sen että kaupunki – postinumero pareja toistui turhan paljon ja oli myös normalisoinnin kannalta parempi ratkaisu tehdä kaupungeista oma taulunsa. Ohjelmallisen toteutuksen aikana törmättiin ongelmaan, jossa esineiden lisääminen pakettiin vaatisi esineen luomisen uudestaan sitä lisättäessä pakettiin. Tämä ei olisi ongelma, mikäli paketissa olisi vain yksi kutakin esinettä, mutta samassa paketissa voi olla useita samoja esineitä, ne on pystyttävä tunnistamaan toisistaan. Tästä johtuen loin itemType taulun, jonka sisältö vastaa käyttäjälle näytettävää esinelistaa. Näistä tyypeistä sitten luodaan varsinaiset esineet joilla on ID ja pakettiID, missä paketissa ne ovat. SmartPostille päädyttiin tekemään ID, vaikka aluksi leveys- ja pituuspiiri yhdessä muodostivat

Näkymät ja ”kiinteät” tiedot: Hyväksi havaitut näkymät, jotka luotiin muun muassa helpottamaan kyselyitä tietokantaan ja tietojen seuraamisen helpottamista ovat:

(1) smartPostView, yhdistetään automaattiin sen kaupunki, postinumeron avulla.
(2) itemTypeView, yhdistetään esinetyyppi ja sen särkymistyyppi eli onko se normaali, särkevä, suojaava vai erikoisesine.
(3) packageView on näkymä, jossa nähdään paketin kuljetus, kuljetusluokka sekä tarkat tiedot kuljetuksen lähtö- ja kohdeautomaateista.
(4) inPackagesView on näkymä, jossa näkyvät kaikki eri paketeissa olevat esineet listattuna.
(5) timoteiManView, yhdistetään TIMOTEI-mies hänen stressinpurku toimenpiteisiinsä.

Tietokannassa on kiinteästi ja käyttäjän, muuttamattomissa kolme kuljetusluokkaa, neljä esineen särkymis/särkemis luokkaa. Viisi TIMOTEI-miestä, joita ei suunnitelmasta huolimatta voi poistaa tai luoda lisää ohjelman käyttöliittymässä. Toinen muutettu ominaisuus oli se, ettei XML:stä saatuja SmartPostien aukioloaikoja käytetty muuhun kuin

infotekstiksi. Alun perin oli tarkoitus mahdollistaa pakettien lähetys vain jos automaatti oli auki ja matkaan kuluisi jokin määrä aikaa. Index.html:sta saatiin kuitenkin vain matkan etäisyys ja monet aukioloajat olivat hyvin monimutkaisia ja riippuvaisia päivistä ja jopa vuodenajasta. Tästä syystä muutin delivers-taulun ainoaksi omaksi attribuutiksi kuljetuksen matkan eli etäisyyden lähtö- ja kohdeautomaatin välillä, jonka sai Googlen APIsta melko kivuttomalla indexin muutoksella.

Huomio tehtävänannosta: Tehtävänannossa annettiin lisäpisteitä (javan osalta) mahdollisesta pitkäaikaisesta lokista, jossa olisi kaikkien käyttökertojen tilastot. Tällainen olisi helppo toteuttaa luomalla tilastotaulu tietokantaan. Se ei kuitenkaan olisi liitoksissa muihin tauluihin, vaan olisi relaatiomallin vastaista, joten päätin olla tekemättä tällaista taulua. Myöskin tilastojen kirjoitus tiedostoon vaikutti huonolta ratkaisulta, kun kyseessä on käytössä tietokanta, joka on tarkoitettu tietojen tallentamiseen. Näistä syistä en tehnyt lainkaan pitkäaikaistilastoa.

Testatessa huomioitavaa: Tietokannan luon on tarkoitus tehdä siten, että käytetään .read-komentoa ja lukea schema- tiedosto, joka myös avaa tietokannan harkkakanta.db nimellä. (Huomasin että tehtävänannossa pyydetään .sqlite tiedostoa, mutta en huomannut mitään eroa .db ja .sqlite toiminnoissa. En muuttanut sitä ettei vahingossa johonkin jää .db .sqlite sijaan ja jokin rikkoutuisi silloin ohjelmassa.) Lisäksi ohjelma hakee tietokannan Netbeansin projektikansiosta, ei dokumenttikansiosta, jossa tietokanta palautetaan tehtävänannon mukaan.

Koodissa huomioitavaa: Suurin osa eheyssäännöistä toteutetaan CHECK –määrityksillä, jotka on kerrottu aiemmassa osiossa. Joitain kohtia tarkistetaan, myös ohjelmassa, jotta vältetään SQLite –virheet. Lisäpisteisiin edellyttävä neljän JOINin kysely ei sellaisenaan toteudu missään, mutta tein Viewin, jossa on kolme JOINia ja yhdistin sen toiseen Viewiin, joka koottiin kahdesta eri JOINista. Näin jälkimmäinen Viewi toteutetaan lopulta viiden eri JOINin avulla. SELECT- kyselyt ovat tarkoituksella mahdollisimman yksinkertaisia ja Viewit ovat monimutkaisempia, mutta niihin on helppo tehdä yksinkertaisia kyselyitä.

4 KESKUSTELU

Voisin keskustella paljonkin viivytyksistä, joita koin alkuperäisen DL:n tienoilla. Ne eivät kuitenkaan millään tapaa enää muuta itse ohjelmaa tai missään vaiheessa vaikuttaneet tietokantatoteutukseen, joten totean vain että tein tämä nyt myöhässä, mutta tein kuitenkin, mikä on mielestäni pääasia.

Tehtävänantoon ja osien laajuuteen liittyvänä huomiona sanoisin sen että, mielestäni tietokantaosuus oli todella pieni verrattuna ohjelmointi osuuteen ainakin käyttämäni ajan perusteella. Tehtävänannossa kuvat vaiheet 1 ja 2 tein molemmat saman päivän aikana. Tietokannan muuttamiseen tai siihen tehtävien kyselyiden suunnitteluun java-osuuden aikana olivat yhteensäkin vain muutamia tunteja kun taas ohjelmalliseen suunnitteluun ja toteutukseen meni viikkoja. (Johtunee myös hitaasta ohjelmointi tahdistani.)

Muutama matkan varrella mieleen tullut kysymys:

(1) Mitä tarkoittaa tehtävänannossa kohta: ”Tietokanta on testattavissa ja käytettävissä sekä SQLite3 että Java-sovelluskäyttöliittymän kautta.”
Tarkoittaako se että käyttöliittymässä tulisi olla kohta, jossa voidaan suoraan syöttää tietokantaan SQL-lauseita?

(2) Olisiko tilastotaulun tekeminen kuitenkin sallittua, sillä tavallaanhan tilastot ovat liitoksissa jokaiseen muuhun tauluun?

(3) Tukeeko SQL-kieli Arrayta, mutta SQLite ei, koska näin ymmärsin pitkähkön googlailun jälkeen. Tästä jatkokysymyksenä onko esimerkiksi koon tallentaminen [x, y, z] Arrayksi hyvä relaatiomallin kannalta. Kuitenkin kyseessä on tietokannassa vain yksi Array yhdellä rivillä, mutta sen sisällä taas on monta arvoa, mikä tavallaan rikkoo suunnitteluperiaatetta.