Tommi Vehviläinen

Relaatiotietokannat ja olio-ohjelmointi

Tietotekniikan kandidaatintutkielma

23. tammikuuta 2017

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Tommi Vehviläinen

Yhteystiedot: tommi.r.o.vehvilainen@student.jyu.fi

Ohjaajat: Sanna Mönkölä ja Sanna Juutinen

Työn nimi: Relaatiotietokannat ja olio-ohjelmointi

Title in English: Relational databases and object oriented programming

Työ: Kandidaatintutkielma

Sivumäärä: 23+0

Tiivistelmä: Olioparadigma on yleisesti hyväksytty tapa mallintaa sovellusohjelman monimutkaisuutta. Vastaavasti relaatioparadigmaan perustuvat relaatiokannat ovat keskeisessä roolissa tiedon tallentamisessa. Näiden kahden paradigman yhteensovittaminen ei ole ongelmatonta, vaan johtaa olio-relaatioyhteensopimattomuutena tunnettuun ongelmaan. Tämä tutkielma tarkastelee yhteensopimattomuuteen johtavia syitä ja esittelee metodeja sen kanssa selviytymiseksi.

Avainsanat: olio-ohjelmointi, relaatiotietokannat, SQL, olio-relaatioyhteensopimattomuus

Abstract: Object-oriented paradigm is a widely accepted way to control complexity in software. Correspondingly the relational databases based on relational paradigm play a central role in storing information. Combining these two paradigm is not effortless but leads to a problem known as the object-relational impedance mismatch. This study examines the causes of the mismatch and presents methods to live with it.

Keywords: object-oriented programming, relational databases, SQL, object-relational impedance mismatch

Sisältö

1 Johdanto

Olio-ohjelmointi on pitkään ollut valtavirran hyväksymä tapa mallintaa ongelmia. Vastaavasti relaatiotietokannat ovat olleet pitkään lähes standardi digitaalisen tiedon tallentamisessa. Olio-ohjelmointi on suunniteltu ensisijaisesti tarjoamaan ohjelmistokehittäjälle ymmärrettäviä ja helposti käsiteltäviä abstraktioita, kun taas relaatiomalli keskittyy isojen tietomäärien tehokkaaseen käsittelyyn relaatioalgebran mukaisilla operaatioilla. Käytännössä oliomalli on monesti haluttu abstraktio monimutkaisuuden hallintaan ja relaatiotietokanta on haluttu tapa tallentaa tietoa. Muita mahdollisia syitä kahden paradigman käyttöön ovat esimerkiksi relaatiotietokantojen hyödyntäminen muissa organisaation toiminnoissa ja taaksepäinyhteensopivuuden säilyttäminen. Tämän vuoksi on tarpeellista kehittää toimivia metodeja näiden kahden paradigman yhdistämiseksi ja vastaavasti näiden metodien määrä ja tunnettavuus voi kannustaa valitsemaan kaksi erillistä paradigmaa.

Ongelman tunnettavuudesta kertoo myös paradigmojen yhteensovittamiseen tarjolla olevien kaupallisten sekä avoimen lähdekoodin ratkaisujen määrä. Nämä ratkaisut vaihtelevat suhteellisen yksinkertaisista relaatiomallia painottavista arkkitehtuurimalleista monimutkaisiin olio-relaatiokuvaustyökaluihin. Tunnettujen ratkaisujen lisäksi kirjallisuudessa on esitetty useita uniikkeja malleja paradigmojen yhteensovittamiseksi. Eräs aiempi oliokeskeinen ratkaisuehdotus yhteensopimattomuusongelman ratkaisemiseksi on ollut relaatiotietokantojen laajentaminen olio-ominaisuuksilla, mutta tällaiset olio-relaatiokannat eivät ole saavuttaneet suurta suosiota. Relaatiokantojen vahvasta asemasta johtuen yhteensovitus pyritään usein tekemään sovellusohjelmassa.

Tämän tutkimuksen aiheena on tarkastella relaatio- ja oliomallin yhteensovittamiseen liittyviä ongelmia sekä esitellä käytössä olevia ja ehdotettuja malleja näiden ongelmien ratkaisemiseksi. Luvussa 2 käydään läpi tarkemmin sellaisia paradigmojen erityispiirteitä, jotka ovat erityisen tärkeitä tämän tarkastelun suorittamiseksi. Tämän jälkeen luvussa 3 esitellään yhteensopimattomuuteen johtavia tekijöitä ja tutkitaan tarkemmin muutamaa yhteensopimattomuuden ilmentymää. Lopuksi luvussa 4 tarkastellaan käytännönläheisestä näkökulmasta relaatiotietokannan ja oliopohjaisen sovellusohjelman yhteensovittamista, tietokantaabstraktion mahdollisia tehtäviä ja metodeja eriasteisten olio-relaatiokuvausten toteuttami-

seksi.

Huolimatta viimeaikaisesta kehityksestä NoSQL-tietokantojen parissa sekä funktionaalisten ja moniparadigmaisten kielten suosion noususta, uskon alussa mainittujen syiden johtavan tulevaisuudessakin olio-relaatioyhteensopimattomuusongelman ajankohtaisuuteen. Tämän aiheen valintaan on johtanut ongelman pysyvyys ja kiinnostus paradigmojen eroista seuraaviin ongelmiin, jotka ovat usein läsnä ohjelmistokehitysprosessin yhteydessä.

2 Kaksi paradigmaa

Yhteensopimattomuusongelmaan johtavien syiden ymmärtämiseksi ja niiden vaikutuksien arvioimiseksi on oleellista tutustua ensin relaatio- ja olioparadigman keskeisiin konsepteihin. Tässä luvussa käydään läpi näiden mallien oleellisia piirteitä.

2.1 Relaatioparadigma

Nykyisten relaatiotietokantojen teoreettisena perustana on Codd (1970) esittämä relaatioteoria. Coddin malli perustuu tietoa sisältäviin relaatioihin ja relaatioalgebran mukaisiin operaatioihin, joiden avulla tiedolla voidaan vastata kysymyksiin. Seuraava relaatiomallin ydinkonseptien esittely pohjautuu teokseen Date 2006.

- **Monikko** t on joukko <Ai, Ti, vi> järjestettyjä pareja, jossa Ai on attribuutin nimi, Ti attribuutin tyyppi ja vi on arvo tyyppiä Ti.
- Järjestetty pari <Ai, Ti> on monikon t **attribuutti** ja se on yksilöllisesti tunnistettavissa attribuutin nimestä Ai. Arvo vi on attribuutin arvo attribuutin nimelle Ai relaatiossa t. Tyyppi Ti on vastaavan attribuutin tyyppi.
- Kaikkien monikon attribuuttien joukko on **otsake**.
- **Relaatio** r koostuu sisällöstä ja otsakkeesta. Otsake on yllä määritelty monikon otsake. Relaatiolla on samat attribuutit ja asteluku kuin otsakkeella. Sisältö on joukko monikoita, joilla kaikilla on sama otsake. Relaation kardinaliteetti on monikoiden joukon koko.

Relaatiokannan tietokantataulut vastaavat relaatiomallin relaatioita ja taulun rivit vastaavat monikoita. Tauluihin tallennetaan relaation totuusväitettä vastaavia rivejä. Valitun näkökulman kannalta relaatiokantojen keskeinen ominaisuus on kuitenkin sen pohjautuminen hyvin määriteltyyn matemaattiseen teoriaan.

2.2 Olioparadigma

Koska ohjelmistosuunnittelu on nimensä mukaisesti suunnittelua eikä vain toteutusta, on tarve luoda malleja joilla tätä suunnittelutyötä voidaan jakaa pienemmiksi ratkaistaviksi osakokonaisuuksiksi. Olioparadigma vastaa tähän tarpeeseen tarjoamalla mahdollisuuden luoda ymmärrettäviä abstraktioita. Tällöin suunnittelussa sekä myöhemmin toteutuksessa ja ylläpidossa voidaan keskittyä rajattuun määrään toiminnallisuuksia halutulla abstraktiotasolla.

Toisin kuin relaatiotietokannoilla, olio-ohjelmoinnin taustalla ei ole täsmällisesti määriteltyä teoriaa. Käytännössä kielen luokittelu oliokieleksi ei usein kerro paljoakaan kielen tarjoamista ominaisuuksista. Esimerkiksi Smalltalk- ja Ruby-ohjelmointikielet koostuvat vain olioista, kun taas monet muut oliokielet sallivat myös primitiiviarvoja. Jotkin kielet kuten Scala, JavaScript ja Python mahdollistavat olioiden lisäksi myös muiden paradigmojen hyödyntämisen. Kielen ominaisuuksien lisäksi tulee olio-relaatioyhteensopimattomuutta pohtiessa ottaa huomioon kielen tyypillisimmät käyttötarkoitukset sekä yhteisössä syntyneet idiomit.

Booch (2006) määrittelee olio-ohjelmoinnin toteutuksen metodiksi, jossa ohjelmat koostuvat yhdessä toimivista joukoista olioita, joista jokainen on jonkun luokan instanssi ja jonka luokat kuuluvat luokkien hierarkiaan perinnän kautta. McConnell (2004) nimeää ohjelman monimutkaisuuden vähentämisen riittäväksi syyksi luoda olion. Nämä määritelmät antavat itse kielen toteutukselle sekä ohjelmoijan luomille abstraktioille runsaasti liikkumavaraa. Ensimmäinen määritelmä ei myöskään huomio prototyyppipohjaisia olioita, jotka ovat vaihtoehto tyypillisimmille luokkaan pohjautuville olioille. Heikosta määritelmästä johtuen kuvattaessa relaatiomuotoista tietoa oliomuotoiseksi, joudutaan keskittymään joukko-opin mielessä eräänlaiseen oliokielten ominaisuuksien leikkaukseen. Seuraavaksi käydään läpi niitä ominaisuuksia, joita tällaiselta oliokieleltä voidaan odottaa (Fussel 1997), (Booch 2006).

- Jokaisella oliolla on varsinaisesta tilastaan poikkeava **identiteetti**, jonka avulla se on yksiselitteisesti tunnistettavissa. Tämä identiteetti on erillinen olion tilasta, joten samassakin tilassa olevat oliot ovat erotettavissa toisistaan.
- Koska oliolla on identiteetti, on mahdollista seurata olion arvojen muutosta ja on mielekästä puhua olion **tilasta**. Tarkemmin sanottuna olion tilaan kuuluu identiteettiin liittyvän olion attribuuttien arvot. Koska oliot eristävät tietoa, olio on abstraktio sisältä-

mästään tilasta joka paljastuu vain olion käyttäytymistä tarkastelemalla.

- Olion **käyttäytyminen** on olion tarjoamien operaatioiden, näiden kutsujen kutsujalleen antamien vastausten ja kutsujen aiheuttamien olioon tai muihin olioihin kohdistuneiden muutosten joukko. Kaiken olioon kohdistuvan vuorovaikutuksen tulee tapahtua olion julkisen rajapinnan kautta ja olion tilasta tulee saada tietoa vain sen käyttäytymisen kautta.
- **Kapselointi** estää ulkopuolisia tahoja näkemästä olion tilan tai toiminnallisuuden toteutusta ja estää näitä luottamasta siihen omassa toteutuksessaan. Olioiden käyttäjät voivat olla vuorovaikutuksessa olioon vain sen julkisen rajapinnan kautta.
- **Tyyppi** on spesifikaatio rajapinnasta, jota oliot voivat tukea. Olio toteuttaa tyypin, jos sen rajapinta tarjoaa tyypin vaatiman toiminnallisuuden. Kaikkia samaa tyyppiä olevia olioita voi käyttää saman rajapinnan kautta. Yksi olio voi toteuttaa useamman tyypin.
- Tyypit voivat olla **assosiaatio**-suhteessa toisten tyyppien kanssa, joka määrää että yhdentyyppiset oliot voidaan linkittää toisentyyppisiin olioihin. Tällaisella linkillä tarkoitetaan mahdollisuutta päästä käsiksi toisen olion julkisen rajapinnan tarjoamaan toiminnallisuuteen ensimmäisen olion kautta.
- Eräs mahdollisuus toteuttaa olio on **luokka**, joka määrittelee usean olion toiminnallisuuden. Luokka määrittelee olion toteuttamat tyypit, rajapintojen vaatiman toiminnallisuuden toteutuksen ja sen kuinka olion tilaa ylläpidetään. Tällöin kukin olio vastaa vain omasta tilasta ja toiminnallisuudestaan. Vaikka luokkien käyttäminen on yleisin lähestymistapa olioihin, se ei kuitenkaan ole ainoa mahdollisuus.
- **Perintä** voi koskea tyyppejä tai luokkia. Kun se koskee tyyppejä ja olio toteuttaa tyypin B siten että tyyppi B perityy tyypistä A, voidaan tätä oliota käyttää myös tyypin A tavoin. Kun perintä koskee luokkia, perintä määrittelee luokan käyttävän perittävän luokan toteutusta mahdollisilla ylikirjoituksilla.

3 Olio-relaatioyhteensopimattomuus

Olio-relaatioyhteensopimattomuus jakaa vahvasti mielipiteitä: osa näkee relaatioiden ja luokkien sekä luokkien attribuuttien ja relaatioiden attribuuttien välillä selvän yhteyden ja itse ongelmaksi jää vain määritellä yhtenäinen terminologia ja kuvauksia tiettyihin erityistapauksiin. Esimerkiksi (Fussel 1997) nimeää paradigmojen yhdistämistä helpottavaksi tekijäksi relaatioparadigman keskittymisen itse tietoon ja olioparadigman keskittymisen ensisijaisesti käyttäytymiseen. Fussellin ehdottama ratkaisu kuitenkin vaatii relaatiotietokannan laajentamista olio-ominaisuuksilla tai näiden muutosten emulointia olemassaolevalla teknologialla. Tiedon tallentaminen ja esittäminen uudessa muodossa herättää kuitenkin olennaisen kysymyksen siitä, täyttääkö tietokanta edelleen ne vaatimukset joiden perusteella projektissa on päädytty relaatiokantoihin. Olio-relaatioyhteensopimattomuuden suurempana ongelmana näkevät tuovat esille lähtökohtaisesti eri tarkoituksiin suunnitellut paradigmat ja niiden eroista johtuvat useat yhteensopimattomuudet. Muista näkökulmista poikkeavana mainittakoon erikseen Meijer (2006), joka esittelee kieleen integroidut kyselyt (engl. *Language Integrated Query, LINQ*) ja väittää välttävänsä yhteensopimattomuuden täysin keskittymällä tietomallien operaatioiden yhtenäisyyksiin.

Olio-relaatioyhteensopimattomuutta käsitellessä tulee huomioida myös ongelman monimuotoisuus. Kaikki kompleksisuus ei johdu vain kahden paradigman yhteensovittamisesta, vaan ongelman ratkaisuun suunniteltu ORM-työkalu (engl. *Object Relational Mapping framework*) joutuu ratkaisemaan myös muita ongelmia. Tyypillisesti tietokanta toimii erillisellä palvelimella, jolle useat ohjelmat voivat lähettää tietokantakyselyitä. Olio-relaatiokuvauksen lisäksi ORM-työkalu joutuu siis hallitsemaan mahdollista välimuistia sekä sovellusohjelman ja tietokantapalvelimen välistä viestintää, josta puhutaan tarkemmin luvussa 4. Fowler (2012) nostaa esille myös relaation ja olion välisen kuvauksen olevan osa yleisempää ongelmaa muuttaa joustava, muistissa käsiteltävä, tietotyyppi relaatiomallin mukaiseksi tietotyypiksi. Tämän lisäksi luvussa 3.1.3 on jätetty käsittelemättä kaksoisskeemaongelma (engl. dual schema problem) joka ei ole spesifi olio-relaatioyhteensopimattomuudelle, vaan koskee kaikkia kuvauksia määrätyn tietorakenteen ja skeemallisen tietovaraston välillä.

Kaupallisten ratkaisujen ja kirjallisuudessa esitettyjen ratkaisuehdotusten määrä ja erilai-

set lähestymistavat viestivät vahvasti universaalin toimivan ratkaisun puuttumista sekä yhteensopimattomuusongelman sivuuttamattomuudesta. Ireland ym. (2009) tarkastelee olio-relaatioyhteensopimattomuutta tarkemmin jakaen sen paradigmojen ja kielten eroista johtuviin ongelmiin sekä eroihin skeema- ja instanssitasoilla. Seuraavaksi tarkastellaan yhteensopimattomuutta yleisellä tasolla Irelandin esittelemän viitekehyksen mukaan. Tämän jälkeen tarkastellaan muutamaa tarkemmin rajattua yhteensopimattomuusongelmaa ratkaisuvaihtoehtoineen kuten ne on esitetty paljon käytetyn Hibernate-kirjaston toimintaa kuvaavassa kirjassa Bauer ja King (2005). Hibernate on ORM-työkalu joka pyrkii mahdollistamaan olioiden käytön tietokannan käsittelyssä. Yhteensopimattomuutta käsiteltäessä esitellään siis myös lukuun 4.4 liittyviä metodeja ja ongelmia.

3.1 Yhteensopimattomuuden erittely

3.1.1 Paradigma

Vaikka oliokielillä ei ole yhtenäistä terminologiaa tai määritelmää, perustuvat ne aiemmin mainittuihin konsepteihin: luokkiin, aliluokkiin, olioihin, attribuutteihin ja assosiaatioihin. Sen sijaan relaatiomallille on täsmällinen määritelmä eikä siis ole epäselvyyttä siitä mitä relaatio tai monikko tarkoittaa tai kuinka sitä käytetään.

Relaatio esittää todellisuuden jotain ilmiötä ja relaation monikko on tähän kohdealueeseen liittyvä totuusväite. Näille konsepteille ei ole suoraa vastinetta olioparadigmassa. Lisäksi oliolla on tilasta erillinen identiteetti ja se kapseloi tilansa toisin kuin monikko tai tietokantataulu. Luokka määrittele hyväksyttävät attribuutit ja olion käytöksen, mutta se ei perustu predikaattilogiikkaan. Relaatiomalli perustuu pohjimmiltaan vain totuusväitteisiin, mutta olioparadigmalle ei ole vastaavaa semantiikkaa.

Olion käyttäytymisen määrittelee sen metodit. Monikoiden voidaan ajatella olevan yhteensopivia tässä suhteessa, koska ne eivät ota kantaa käyttäytymiseen. Sen sijaan relaatioille voidaan suorittaa relaatioalgebran mukaisia operaatioita kuten projektioita, rajoittamisia tai yhdisteitä. Yllämainitut eriäväisyydet johtuvat pohjimmiltaan paradigmojen erosta ja luokitellaan konseptuaalisen yhteensopimattomuuden (engl. *conceptual mismatch*) alle.

3.1.2 Kieli

Jokainen kieli heijastaa sitä paradigmaa johon se perustuu. Oliokielet perustuvat joukkoon määriteltyjä luokkia, jotka voidaan nähdä ohjeina ajonaikaisten olioiden toiminnalle. Vastaavasti SQL-kielellä kuvattu tietokantaskeema on kuvaus relaatiomallin relaatioita vastaavista tietokantatauluista. Tietokantataulun rivin semantiikka ei ole yhtä formaali kuin monikolla, vaan se esittää tietoa jostain kohdealueen asiasta. Tässä tapauksessa paradigma heijastuu kieliin siten että SQL-haku kerää tietoa valituista tauluista vastaten kysymykseen. Vastaavasti oliokielessä oliolle annetaan tehtävä, jonka osia se delegoi eteenpäin muille olioille niiden vastuiden mukaan.

Myös tyyppien merkitys on keskeinen ero kielissä. Oliokielissä luokat tyypillisesti laajentavat ohjelman sallittujen tyyppien joukkoa, mutta SQL-kielessä luokat eivät ole laajennuksia tyyppijärjestelmään eikä oliokielen vastaavaa luokkaa voi käyttää esimerkiksi taulun sarakkeen tyyppinä. Nämä kielestä johtuvat yhteensopimattomuusongelmat luokitellaan kuvauksellisen yhteensopimattomuuden (engl. *representational mismatch*) alle.

3.1.3 Skeemataso

Yhteensopimattomuutta ei aiheuta vain eri kieli ja eri käsitteet, vaan myös syy luoda uusi luokka poikkeaa syystä luoda uusi tietokantataulu. Oliomalli keskittyy jakamaan keskenään vuorovaikuttavat oliot datan ja käyttäytymisen mukaan ymmärrettäviksi ja helposti käytettäviksi abstraktioiksi. Relaatiomalli sen sijaan keskittyy suurten tietomäärien tehokkaaseen käsittelyyn, tiedon yhtenäisyyden säilyttämiseen ja turhan tiedon tallentamisen välttämiseen. Tämä yhteensopimattomuuden muoto on painotuksellinen yhteensopimattomuus (engl. *emphasis mismatch*), joka seuraa siitä että paradigmat on suunniteltu ratkaisuiksi eri ongelmiin. Tästä syystä samaa ongelmaa mallinnettaessa näillä kahdella paradigmalla päädytään helposti kahteen erilaiseen ratkaisuun.

Skeemalla tarkoitetaan tiedon staattista rakennetta, joka tässä kontekstissa ilmenee tietokannan skeemana tai oliokielen luokkina. Skeematason suunnittelussa paradigmojen välillä on aikaisemmin mainitun loogisen jaon lisäksi konkreettisempiakin rajoitteita, kuten relaatiomallin ensimmäinen normaalimuoto. Ensimmäinen normaalimuoto kieltää moniarvoiset att-

ribuutit kuten listat ja joukot (Kent 1983). Käytännössä skeematason ongelmat tulee ratkaista käyttötarkoituksen mukaan yksi- tai kaksisuuntaisilla kuvauksilla.

Tämä yhteensopimattomuuden muoto laajenee teknologialähtöistä näkökulmaa laajemmalle: mikäli organisaatiossa eri osastot vastaavat kehitystyöstä ja tietohallinnosta, on ongelmana päättää kuka vastaa tietokantaskeeman suunnittelusta ja millä perusteella se tehdään. Tässä tapauksessa myös skeeman muuttaminen refaktoroinnin yhteydessä tai tiedon hakeminen olioparadigman ehdolla suunnitellusta tietokantaskeemasta voi olla haastavaa. Myös näiden ongelmien voidaan nähdä kantautuvan pohjimmiltaan olio-relaatioyhteensopimattomuudesta.

3.1.4 Instanssitaso

Instanssiyhteensopimattomuuden (engl. *instance mismatch*) perimmäinen syy on olioiden käsittely atomisina yksiköinä, vaikka ne todellisuudessane voidaan jakaa rakenteeseen, tilaan ja käyttäytymiseen. Olion rakenne kuvataan sekä olion määrittelevässä luokassa että SQL-skeemassa. Luokassa määritellään myös olion käytös. Tilaa tulee ylläpitää muistissa olevassa oliossa sekä tietokannan taulun rivissä, jossa olion tila voi olla kuvattuna yhtenä tai useampana rivinä. Tämä tiedon jakaminen rikkoo olioparadigman abstraktion ja aiheuttaa usein turhan tiedon hakemista sovelluksissa jotka eivät käytä kaikkea tallennettua tietoa. Käyttäytyminen määritellään vain oliossa. Olioita, tai tarkemmin oliot määrittteleviä luokkia, voidaan siis suunnitella toiminnallisuuden perusteella toisin kuin tietokantarivejä. Tämä havainto tukee aiempaa väitettä olion ja tietokantarivin luomisen erilaisista syistä.

3.2 Rakeisuus

Rakeisuudella (engl. *granularity*) viitataan käsiteltävien olioiden suhteelliseen kokoon, johon liittyvä ongelma on erilaisten olioiden kuvaaminen vastaaviksi tietokantatauluiksi. Esimerkiksi mallinnettaessa kohteita käyttäjä ja osoite voi oliokielessä rakeisuuden asteita olla tyypit käyttäjä sekä osoite ja näiden sisältämät primitiiviarvoista koostuvat attribuutit. Samaa kohdetta voidaan mallintaa tietokannassa yhtenä tauluna, jossa rakeisuutta on pienennetty sisällyttämällä tauluun käyttäjät myös osoitteen tiedon primitiiviarvoina.

Bauer ja King (2005) nimeää rakeisuudesta johtuvat ongelmat helposti ratkaistaviksi, mutta olioparadigman abstraktion kannalta tärkeiksi. Esiintymistiheudestä johtuen sopivan kuvauksen löytäminen on tärkeää myös suorituskyvyn kannalta. Intuitiivisesti luokan attribuutit vastaavat tietokantataulun attribuutteja ja sopiva kuvaus saataisiin kuvaamalla relaatiot luokiksi ja merkitsemällä assosiaatiot viiteavaimella. Tietokantataulujen yhdistämiseen käytettävien liitosoperaatioiden ajallisesta kalleudesta johtuen tällainen kuvaus ei kuitenkaan todennäköisesti olisi optimaalinen. Myös instanssiyhteensopimattomuuden käsittelyn yhteydessä esille tulleet eri syyt luoda olioita ja relaatioita ovat nähtävillä: tietokannassa pelkillä osoitetiedoilla ei ole suurtakaan käyttöarvoa, mutta oliokielessä osoitteen toiminnallisuus saadaan eroteltua käyttäjästä. Rakeisuuden muuttaminen paradigman mukaan vähentää liitosoperaatioita, mutta toisaalta voi johtaa ylimääräisen tiedon hakemiseen (Russell 2008).

3.3 Alityypit

Oliokielissä olioille yhteistä skeemaa kuvataan olioiden välisellä perintäsuhteella. Relaatiomalli ei tarjoa vastaavaa toiminnallisuutta, vaan perintä tulee kuvata jollain muulla tavalla. Käytännössä luokkahierarkioiden mallintamiseen on kolme lähestymistapaa: tietokantataulu jokaiselle luokalle, tietokantataulu jokaiselle konkreettiselle luokalle ja luokkahierarkian kuvaaminen yhdeksi tauluksi (Ambler 2000). Myös toiselle olio-ohjelmoinnin tyyppeihin liittyvälle keskeiselle ominaisuudelle, polymorfisuudelle, puuttuu vastine SQL-kielestä. Koska tietokantataulun viiteavain voi viitata vain yhteen tauluun, ei polymorfisten kyselyiden toteuttaminen ole suoraviivaista.

3.4 Identiteetti

Olio- ja relaatiomallin yhdistävä järjestelmä sisältää samalle tiedolle kolme erilaista identiteettiä: oliokielessä identiteetti viitteen ja arvon mukaan sekä tietokannassa rivin perusavaimen perusteella määräytyvä identiteetti. Ei ole tavatonta että ohjelmassa on samanaikaisesti useita tietokantarivin perusavaimen kannalta saman identiteetin omaavia olioita esimerkiksi monisäikeisyyden johdosta. Tällaisessä tilanteessa oliokielen arvo- eikä viiteidentiteetti yhdistä olioita, joten identiteetin käsittely vaatii erityistä huomiota. Toinen identiteettiin liit-

tyvä ongelma on tietokannassa: esimerkiksi taulun käyttäjä rivin yksilöi yleensä palvelussa käytetty käyttäjänimi tai sähköpostiosoite. Tähän perusavaimeen kuitenkin viitataan mahdollisesti muista tauluista, joten perusavaimena olevan arvon muuttaminen on ongelmallista. Mahdollisesti pitkänkin perusavaimen, esimerkiksi sähköpostiosoitteen, toistaminen jatkuvasti vie myös ylimääräistä tallennustilaa. Tästä ongelmasta päästään eroon käyttämällä tietokannassa rivin identiteettinä synteettistä perusavainta, joka ei ole riippuvainen rivin sisällöstä.

Olioiden muodostama graafi mahdollistaa sen, että usea kaari johtaa samaan olioon. Tällaista graafia muodostettaessa identiteetin väärä käsittely voi aiheuttaa kahden instanssin luonnin samasta oliosta. Tämä luonnollisesti johtaa olion väärään käyttäytymiseen ja ohjelman ei-toivottuun toimintaan (Keith ja Stafford 2008).

3.5 Assosiaatiot

Mallinnettavan kohteen assosiaatiot kuvaavat yksiköiden välisiä suhteita. Olio-ohjelmassa assosiaatioita kuvataan olioviitteillä ja relaatiomallissa käytetään viiteavaimia. Olioviitteet ovat luontaisesti yksisuuntaisia: viite on yhdestä oliosta toiseen eikä viitattava olio ole tietoinen assosiaatiosta. Jos viitteestä halutaan kaksisuuntainen, täytyy assosiaatio määritellä molemmissa olioissa. Toisaalta relaatiomallissa assosiaatiosta ei voida puhua yksi- eikä kaksisuuntaisena, sillä liitos- ja projektio-operaatioiden avulla voidaan luoda mielivaltaisia assosiaatioita taulujen välille. Assosiaatoiden näkökulmasta ongelmana on siis kuvata kaiken tiedon paljastava tietomalli sovellusohjelman vaatimaan assosiaatioita rajoittavaan muotoon. Tämän lisäksi oliokielet sallivat myös monesta moneen -assosiaatiota, mutta relaatiomallissa ei ole mahdollista tallentaa useaa viiteavainta assosiaatiota kuvaavaan sarakkeeseen. Monesta moneen -liitokset tehdään relaatiomallissa erillisellä linkkitaululla, joka ei kuvaudu oliomalliin. Tällainen linkkitaulu koostuu kahden taulun viiteavaimien pareista mahdollistaen myös monesta moneen -kuvaukset.

3.6 Tiedon hakeminen

Relaatio- ja oliomallin välillä on myös perusteellinen ero tiedon hakemisessa tietorakenteesta. Olioihin perustuvassa kielessä luonnollinen tapa hakea tietoa on kutsua toisen olion palauttavaa metodia. Tämän jälkeen tiedonhakua voidaan jatkaa käyttäen tämän uuden olion julkista rajapintaa. Tässä tapauksessa ohjelman olioiden voidaan nähdä muodostavan verkko, jossa solmut sisältävät olioiden tiedon ja tiedonhaku on assosiaatioita vastaavia kaaria pitkin siirtymistä oliosta toiseen. Vastaava strategia ei kuitenkaan toimi relaatiomallin kanssa.

Tietokantojen suorituskyvyn kannalta tärkeintä on tietokantahakujen määrän minimoiminen. Tämän vuoksi tulokset koostuvat usein toisiinsa liitetyistä tauluista, joihin on pyritty hakemaan haluttu tieto. Tietokantakyselyä muodostettaessa tulisi siis tietää mitä tiedolla aiotaan tehdä. Tämä ei usein kuitenkaan ole mahdollista. Tiedonhakuun on kaksi lähestymistapaa: ahne haku hakee tietokannasta koko oliograafin ja vastaavasti laiska haku noutaa vain pyydetyn tiedon tehden uuden kyselyn assosiaatioiden yhteydessä. Kumpikaan näistä vaihtoehdoista ei ole suorituskyvyn kannalta optimaalinen, vaan sopiva ratkaisu joudutaan etsimään tapauskohtaisesti.

4 Relaatiotietokannat ohjelmistokehityksessä

Aikaisemmin kuvattujen yhteensopivuusongelmien perusteella yleiskäyttöiselle työkalulle paradigmojen yhteistyötä helpottamaan on tarvetta. Ennen tällaisen ohjelman käyttöönottoa tai kehitystyön aloittamista on sille asetettuja vaatimuksia kuitenkin tarkennettava.

Yksinkertaisimmillaan tietokanta-abstraktioita tarjoavat kirjastot tarjoavat isäntäkielen tietokanta-ajuria yksinkertaisemman rajapinnan yleisimpiin käyttötarkoituksiin. Toinen haluttu ominaisuus voi olla SQL-kielen abstrahointi. Jos päätöstä käytettävästä tietokannasta ei haluta tehdä projektin alkuvaiheessa tai kehittäjä joutuu vaihtamaan käyttämäänsä tietokantaa usein, voi tietokantakohtaisen SQL-murteen korvaaminen yleisemmällä kyselykielellä helpottaa kehitystyötä. Esimerkki tällaisesta matalan tason abstraktion tarjoavasta kirjastosta jOOQ, joka mahdollistaa tyyppiturvalliset SQL-kyselyt ja abstrahoi käytetyn SQL-murteen jättäen sopivan kuvauksen löytämisen olio- ja relaatiomallin välille ohjelmistokehittäjän vastuulle (*jOOQ*).

Luvussa 3.6 käsiteltiin tarvittavan tiedon hakemisen vaikeutta. Tapauskohtaiseen optimointiin liittyy vahvasti myös tietokannan rooli sovelluksessa. Tietokanta voi olla usean ohjelman käytössä oleva keskitetty tietovarasto tai se voi olla suunniteltu vain yhdelle sovellusohjelmalle. Jälkimmäisessä tapauksessa tietokantahakujen määrän rajoittamisen tuoma hyöty suorituskykyyn ei ole yhtä merkittävä. Vastaavasti suuren kuormituksen alla toimivaa tietokantapalvelinta käytettäessä operaatioiden suorittaminen välimuistissa voi olla suorituskyvyn kannalta välttämätöntä (Keith ja Stafford 2008).

Jotkin työkalut pyrkivät helpottamaan ohjelmistokehittäjän työtä muuttamalla kyselyiden tulokset olioiksi, jolloin kehittäjä voi työskennellä käyttäen enemmän monimutkaisuuden hallitsemiseen paremmin soveltuvaa olioparadigmaa. Esimerkiksi Hibernate nimeää tavoitteekseen eliminoida 95% manuaalisesta tiedonkäsittelystä SQL-kielen ja JDBC-tietokanta–ajurin avulla (*Hibernate Getting Started Guide*). Tämä on toisaalta selvä signaali siitä, että ohjelmistokehittäjän tulee ymmärtää olio-relaatiokuvauksesta vastaavan ohjelman toimintaperiaate pystyäkseen tekemään jäljelle jäävät 5% kyselyistä matalammalla abstraktiotasolla. Sen sijaan vastaava ORM-työköalu Apache Cayenne listaa yhdeksi syyksi käyttää ohjelmaa

sen ettei se vaadi tietoa SQL:sta, vaikkakin mainitsee että siitä voi olla hyötyä (*Why Cayenne?*). Olioiksi relaatiot kuvaavan ORM-työkalu voi siis jopa mahdollistaa tietokantojen käytön ilman tietoa SQL-kielestä tai tietokannan toiminnasta.

Neward (2006) vertaa esseessään olio-relaatiokuvausta Vietnamin sotaan: alkuvaiheen voitot kannustavat jatkamaan, mutta jäljelle jäävät ongelmat osoittautuvat liian vaikeiksi viimeisen voiton saavuttamiseksi. Aiemmin esitettyjen yhteensopimattomuusongelmien valossa ORM-työkalujen tarjoama abstraktion ei siis voida sanoa olevan eheä, vaan abstraktiota käyttäessä täytyy olla tietoinen kuvauksen toteutuksesta. Puutteellisesta abstraktiosta huolimatta on ohjelman modulaarisuuden ja koodin uudelleenkäytettävyyden kannalta hyödyllistä eristää olio-relaatiokuvauksesta huolehtiminen omaksi yksikökseen.

Relaatiotietokantojen taustalla oleva teoria asettaa tietokannan muodolle tiukat rajoitteet. Oliomalli joustaa tätä enemmän, mutta tarjotakseen lupaamansa helposti käsiteltävän rajapinnan kapseloimaansa tietoon täytyy olion suunnittelun ehtona olla yksinkertaisuuden ja ymmärrettävyyden ylläpitäminen relaatiomallin luomien rajoitteiden sijaan. Seuraavaksi esitellään eritasoisia ohjelmistokehitysprosessiin soveltuvia olio-relaatiokuvauksia (Fussel 1997).

4.1 Puhdas relaatiomalli

Tässä mallissa koko ohjelma käyttöliittymä mukaan lukien perustuu täysin relaatiomuotoisen datan ympärille. Malli mahdollistaa SQL-kyselyn hienosäädön, mutta koodin uudelleenkäytettävyys ja ylläpidettävyys kärsii (Bauer ja King 2005).

Äärimmäinen ratkaisu on luoda käyttöliittymä oliokielellä ja hoitaa tiedon tallentaminen ja liiketoimintalogiikka tietokannassa tallennetuilla proseduureilla. Tällöin tietokantapalvelin toimii keskeisenä osana sovellusta sen sijaan että olisi keskitetty tietovarasto usealle samaa dataa käyttävälle ohjelmalle.

4.2 Kevyt oliokuvaus

Kevyessä oliokuvauksessa entiteetit esitetään luokkina ja muunnos relaatiomallista tehdään manuaalisesti. Tietokantakutsut pyritään eristämään omaan luokkaansa siten että muu ohjelma ei ole suoraan riippuvainen niistä.

Esimerkki tällaisesta kevyestä oliokuvauksesta on Active Record -arkkitehtuurimalli, jossa sovellusohjelman luokat suunnitellaan tietokantataulujen mukaan (Fowler 2002). Tällaiset Active Record -luokat sisältävät toiminnallisuuden sisältämänsä tiedon luomiseksi, poistamiseksi ja muuttamiseksi tietokannassa sekä mahdollistavat tiedon käsittelyn metodeilla sovellusohjelmassa. Mallin pääasiallinen ongelma on sen vaatimus isomorfisesta kuvauksesta tietokantaskeeman ja luokkien välillä mahdollistaen vain yksinkertaisten olio-ominaisuuksien käytön.

4.3 Keskivahva oliokuvaus

Keskivahva oliokuvaus käyttää pääasiassa olioparadigman mukaisia olioita ohjelmakoodissa. Tietokantakyselyt tehdään usein oliopohjaisella kielellä, josta generoidaan vastaava SQL-kysely. Tämän asteisiin ORM-ratkaisuihin liittyy usein erillisen tilan ylläpitäminen muistissa ja tietokannassa välimuistin avulla ja siten mahdollisuus vastata tietohakuihin lokaalisti.

4.4 Vahva oliokuvaus

Vahva oliokuvaus toimii olioparadigman ehdoilla ja sallii kaikkien olio-ominaisuuksien käytön. Myös tämä kuvaus osaa vastata kyselyihin välimuistin perusteella ja lisäksi mahdollistaa useiden yksinkertaisten kyselyiden yhdistämisen yhdeksi monimutkaiseksi tietokannassa suoritettavaksi kyselyksi. Kehittyneet ORM-työkalut tukevat myös erilaisia tiedonhakustrategioita sekä kyselyiden optimointia.

Hienostuneempien olio-ominaisuuksien kuvauksien ongelmiin perehdyttiin luvussa 3 ja ratkaisut näihin kuvausongelmiin on tehtävä tapauskohtaisesti. Koska suurin osa näistä päätöksistä on riippuvaisia sovelluksen käyttötarkoituksesta, tarjoavat monet ORM-työkalut laajat mahdollisuudet konfiguroida esimerkiksi kuvauksien rakeisuutta sekä määritellä tiedonha-

kustrategian oliolle attribuuttikohtaisesti. Nämä konfiguraatiomahdollisuudet antavat viitteitä myös siitä että ohjelmistokehittäjän on tunnettava ORM-työkalun ja tietokannan toiminta saadakseen aikaan suorituskykyisiä kuvauksia.

5 Yhteenveto

Olio- ja relaatioparadigma on selvästi kaksi erilaista tapaa tiedon mallintamiseen ja olio-relaatioyhteensopimattomuus on joukko näiden lähestymistapojen eroista aiheutuvista ristiriidoista ja ongelmista. Ongelmaa on käsitelty laajasti kirjallisuudessa ja kaupallisia sekä avoimen lähdekoodin työkaluja on useita, mutta yksittäistä kaikki ongelmat poistavaa ratkaisua ei ole. Eräs tässä käsittelyssä vähälle huomiolle jäänyt ratkaisu on kieleen integroidut kyselyt, joiden syvällisempi tarkastelu voisi mielenkiintoinen tutkimusaihe.

Yhteensopimattomuusongelmat voidaan jaotella paradigmasta ja kielestä sekä skeema- ja instanssitason yhteensopimattomuudesta johtuviin ongelmiin. Näistä ongelmista huomattavimmat liittyvät tiedon rakeisuuteen, tyyppien merkitykseen, yksikön identiteettiin, assosiaatioihin ja tiedon hakemiseen tietorakenteesta. Näihin ongelmiin pyritään käytännössä vastaamaan eriasteisilla olio-relaatiokuvauksilla. Yhteensopimattomuusongelmiin liittyvässä monimutkaisuudessa tulee kuitenkin huomioida että kaikki asiaan liittyvä kompleksisuus ei ole uniikkia olio-relaatioyhteensopimattomuudelle. Kuvauksesta vastaava ohjelmisto joutuu usein ottamaan kantaa myös välimuistin optimaaliseen toimintaan sekä jaettuna resurssina toimivan tietokantapalvelimen tehokkaaseen käyttöön. Myös tietorakenteiden kuvauksien optimointi voidaan nähdä yhteisenä ongelmana kaikille tietorakenteille, joiden välillä ei ole tunnettuja isomorfisia kuvauksia.

Sekä olio- että relaatiomalli asettavat tiukan rajoitteen tietorakenteelle: relaatiokannat pohjautuvat relaatioteoriaan ja vastaavasti olioparadigma menettää helposti merkityksensä mikäli suunnittelu ei tapahdu olioiden ehdolla. Yksinkertaisia tietorakenteita mallinnettaessa olioparadigman vaatimuksista on helpompi joustaa käyttämällä esimerkiksi ActiveRecord-mallia. Monimutkaisempia oliosuhteita kuvattaessa voi olla tehokkaampaa käyttää vahvempaa oliokuvausta.

Vaikka täysin relaatiomallin abstrahoiva ORM-työkalu on yksinkertaisuudessaan houkutteleva vaihtoehto, se ei välttämättä mahdollista monimutkaisimpien kyselyiden muodostamista. Vahvan kuvauksen heikkoutena on usein myös epäoptimaalisesti generoitu SQL-kysely, joka ei ole riittävän suorituskykyinen aikakriittisissä tilanteissa. Yhteensopimattomuuden kont-

rollointi vaatii työkalun oikeaa konfiguraatiota, joka puolestaan edellyttää ohjelmistokehittäjältä ymmärrystä olio-relaatiokuvauksen periaatteista. Vastaavasti kehittäjän täytyy tunnistaa mahdolliset olio-relaatiokuvaukseen liittyvät pullonkaulat ja aikakriittiset tilanteet, jotka vaativat generoidun tietokantahaun optimointia. Tämän vuoksi ORM-työkalut eivät tee tietokantaosaamista turhaksi, vaan korkeintaan siirtävät usein toistuvan koodin uudelleenkäytettäväksi kirjastoksi.

Lähteet

Ambler, Scott W. 2000. “Mapping objects to relational databases: What you need to know and why”. *Ronin International.*

Why Cayenne? Saataville WWW-muodossa, `https://cayenne.apache.org/why-cayenne.html`. Viitattu 11.12.2016.

Bauer, Christian, ja Gavin King. 2005. “Hibernate in action”.

Booch, Grady. 2006. *Object oriented analysis & design with application.* Pearson Education India.

Codd, Edgar F. 1970. “A relational model of data for large shared data banks”. *Communications of the ACM* 13 (6): 377–387.

Date, Christopher John. 2006. *An introduction to database systems.* Pearson Education India.

Fowler, Martin. 2002. *Patterns of enterprise application architecture.* Addison-Wesley Longman Publishing Co., Inc.

Fussel, Mark. 1997. *Foundations of object-relational mapping.*

Hibernate Getting Started Guide. Saatavilla WWW-muodossa, `https://docs.jboss.org/hibernate/orm/current/quickstart/html_single/`. Viitattu 11.12.2016.

Ireland, Christopher, David Bowers, Michael Newton ja Kevin Waugh. 2009. “A classification of object-relational impedance mismatch”. Teoksessa *Advances in Databases, Knowledge, and Data Applications, 2009. DBKDA’09. First International Conference on,* 36–43. IEEE.

jOOQ. `http://www.jooq.org/`. Viitattu 11.12.2016.

Keith, Michael, ja Randy Stafford. 2008. “Exposing the ORM cache”. *Queue* 6 (3): 38–47.

Kent, William. 1983. “A simple guide to five normal forms in relational database theory”. *Communications of the ACM* 26 (2): 120–125.

McConnell, Steve. 2004. *Code complete.* Pearson Education.

Meijer, Erik. 2006. “There is no impedance mismatch:(language integrated query in Visual Basic 9)”. Teoksessa *OOPSLA Companion,* 710–711.

Fowler, Martin. 2012. *OrmHate.* Saatavilla WWW-muodossa, `http://martinfowler.com/bliki/OrmHate.html`. Viitattu 2.12.2016.

Russell, Craig. 2008. “Bridging the object-relational divide”. *Queue* 6 (3): 18–28.

Neward, Ted. 2006. *The Vietnam of Computer Science.* Saatavilla WWW-muodossa, `http://blogs.tedneward.com/post/the-vietnam-of-computer-science/`. Viitattu 11.12.2016.