Relaatiotietokantojen takaisinmallintaminen

Ari Seppi

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Lokakuu 2004

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Ari Seppi: Relaatiotietokantojen takaisinmallintaminen
Pro gradu -tutkielma, 42 sivua, 24 liitesivua
Lokakuu 2004

Tässä tutkielmassa käsittelen tietokantojen takaisinmallinnusta sekä yleisellä periaatetasolla että käytännön menetelmien kannalta. Työssä esittelen erilaisia kirjallisuudessa esiintyviä takaisinmallinnusmenetelmiä. Tuloksena havaitsin, että menetelmissä on osittain selviä yhtäläisyyksiä, esimerkiksi oliomallin hyväksikäyttö laajemmissa mallinnustavoissa. Lisäksi olen tehnyt oman sovellukseni aiheesta: lisäsin Fujaba-kaavio-ohjelmaan tietokantakaaviotyypin ja siihen erilaisia takaisinmallinnusta helpottavia lisätoimintoja.

Fujaba on lähinnä olio-ohjelmistojen rakentamiseen tarkoitettu työkaluohjelmisto. Alunperin Fujaba on rakennettu UML:n ja Java-kielen välisiä muunnoksia varten siten, että UML-kaavioista on generoitu Java-koodia, mutta myös Java-koodista on voitu takaisinmallintaa UML-kaavioita.

Fujaba-työkalun arkkitehtuuri on sellainen, että siihen on mahdollista toteuttaa uusia toimintoja, kuten uusia kaaviotyyppejä. Olen rakentanut Fujaba-työkaluun tietokantalisukkeen (plug-in). Tietokantalisukkeen avulla voidaan automaattisesti generoida luokkia tietokannan johdonmukaista ja tietokannan kannalta tehokasta käyttöä varten. Lisukkeeseen sisältyy myös takaisinmallinnustoiminto: tietokanta voidaan takaisinmallintaa joko SQL-määrittelytiedostoista tai suoraan JDBC-rajapinnan kautta. Lisukkeella voidaan ehdottaa käyttäjälle vierasavainsuhteita tietokannan rakenteen perusteella silloin, jos näitä vierasavainsuhteita ei esimerkiksi ole mallinnettu itse tietokantaan lainkaan. Lisäksi lisukkeella voidaan tukea tietokantaevoluutiota, eli tietokannan päivittämistä vastaamaan uutta suunniteltua tietokannan kaaviota.

Avainsanat ja -sanonnat: Relaatiotietokannat, takaisinmallintaminen, tietokantakaaviot.
CR-luokat: H 2.1, H 2.4

Sisällysluettelo

1. Johdanto

Takaisinmallinnus (reverse engineering) eli valmiin ohjelmatuotteen toimintatapojen selvittäminen on erittäin tärkeää ohjelmistokehityksessä.

Usein ohjelmistoja tuotettaessa ei aloiteta tyhjästä, vaan lähtökohtana on entinen järjestelmä. Entistä järjestelmää muokataan muuttuneita tarpeita vastaavaksi niin kauan kun se on kannattavaa; muokattu järjestelmä ei yleensä kuitenkaan ole ohjelmistoteknisesti niin hyvä kuin aikaisempi. Ajan myötä muokkaukset kuluttavat järjestelmän muokkauspotentiaalia ja lopulta muokkausten tekeminen vanhaan ohjelmaan käy hankalammaksi ja siten myös liian kalliiksi. Tarvitaan siis kokonaan uusi ohjelmisto, ja sen kehittämiseen takaisinmallinnustekniikoita.

Takaisinmallinnusta tarvitaan, koska entiseen järjestelmään on saattanut sitoutua suuri määrä yritykselle tärkeää hiljaista tietoa. Ohjelma luultavasti toimii yrityksen sisäisten, käytännössä usein tarkemmin määrittelemättömien, käytäntöjen mukaan. Tällainen tieto on tietenkin saatava mahdollisimman täydellisesti talteen ja käyttöön uudessa järjestelmässä, koska sen hukkaaminen aiheuttaisi suuria tappioita.

Entisen järjestelmän dokumentointi saattaa olla puutteellinen tai puuttua kokonaan. Tämä voi johtua järjestelmää rakentaneiden ohjelmoijien ja suunnittelijoiden ajanpuutteesta tai kiireellisistä muutoksista järjestelmässä. Jos muutoksia on tehty ohjelmointivaiheen alkamisen jälkeen, on erityisen suuri mahdollisuus, että muutokset on jätetty dokumentoimatta.

Erityisesti vanhempia järjestelmiä tehtäessä on voitu käyttää epäselviä optimointeja. Aikana, jona koneteho ja muisti olivat erittäin arvokkaita, on koodin selvyyttä uhrattu laskentatehon lisäämiseksi.

Takaisinmallinnusvaiheessa yritetään selvittää, miten ohjelmat toimivat. Tämä on hankalaa, jos sitä joudutaan puutteellisen dokumentoinnin vuoksi tekemään suureksi osin koodin pohjalta. Hankaluudet korostuvat, jos ohjelma on suuri ja ohjelmakoodia paljon.

Ohjelmointikielet ovat ilmaisuvoimaisia ja liikkuvat niin yksinkertaisella tasolla, että täysin oikean kuvan saaminen koodista on toisinaan hyvin vaikeaa. Tietokannat puolestaan toimivat tietovarastona, joten ne ovat yleensä ohjelmia loogisemmin rakennettuja ja niiden tarkoituksen selvittäminen ei ole yhtä hankalaa kuin ohjelmatoiminnaisuuden.

Yleensä saatavilla olevasta tietokannankannan kuvauksesta pääsee selville yleisrakenteesta, ja tietomassoja taas voi analysoida tehokkaasti koneellisesti. Kaikki ei kuitenkaan ole suoraviivaista: tietoalkioiden tarkka merkitys voi helposti jäädä epäselväksi.

Työssäni tulen keskittymään relaatiotietokantaan, koska se on yleisimmin käytetty tietokantatyyppi. Huomion erityisesti oliomallinnukseen liittyvät menetelmät. Tarkoitukseni on siis kartoittaa takaisinmallinnuksen yleistä teoriaa ja sen menetelmiä.

Toisessa luvussa esittelen relaatio- ja SQL-mallien määritelmät, kolmannessa taas annan yleistietoa takaisinmallinnuksen ongelma-alueesta, minkä jälkeen listaan neljännessä luvussa käyttökelpoisia tiedonlähteitä. Viidennessä luvussa käyn läpi tietokannan takaisinmallinnuksen vaiheita ja kuudennessa luvussa suunnittelun yleisiä käytäntöjä siltä osin kuin se on takaisinmallintamisen kannalta olennaista.

Seitsemäs luku esittelee pienempiin yksityiskohtiin keskittyviä takaisinmallinnusratkaisuja, ja kahdeksas kaksi laajempien takaisinmallinnuskäytäntöjen lajittelua. Yhdeksänneksi on vuorossa muutaman tällaisen laajemman käytännön esittely. Lopulta kymmenennessä luvussa esittelen omaa sovellustani, Fujaba-kaavio-ohjelmaan lisäämääni tietokantakaaviotyyppiä, johon voi tuoda kaavioita SQL-tiedostoista tai suoraan tietokannasta ja valmiin kaavion voi muuttaa SQL-lauseiksi tai päivittää suoraan tietokantaan.

2. Määritelmiä

Osa työssä käytetystä kirjallisuudesta perustuu relaatiomalliin, osa taas SQL-malliin. Seuraavassa määrittelen molemmat mallit. Mikäli muuta ei ole sanottu, työssä käytetään Mannilan ja Räihän [1992] tietokantakäsitteitä.

2.1 Relaatiomalli

Relaatiotietokanta R koostuu yhdestä tai useammasta relaatiosta. Jokainen relaatio r koostuu yhdestä tai useammasta attribuutista A_n; tästä käytetään merkintää: r(A_1, A_2...A_n). Jokaisella relaatiolla ja attribuutilla on yksilöivä nimi. Lisäksi jokaisella attribuutilla on arvoalue, jolle kaikki attribuutin saamat arvot sijoittuvat. Attribuutin arvo voi olla myös tyhjä.

Relaatioon r liittyy rivejä, joissa kussakin on n tietoalkiota, yksi jokaista attribuuttia kohden. Relaation sisältö voidaan kuvata taulukkona, jossa attribuuttien nimet ovat sarakkeiden otsikoita ja rivit relaation tietosisältö.

Kahden attribuuttiryhmän r_1 ja r_2 välillä voi olla funktionaalisia riippuvuuksia, jolloin ryhmän r_1 attribuuttien arvot määräävät ryhmän r_2 attribuuttien arvot. Tästä käytetään merkintää r_1 -> r_2. Muodollisemmin määriteltynä: Olkoon R tietokantakaavio ja X $\subseteq$ R, Y $\subseteq$ R. Relaatio r tietokantakaaviossa R sisältää funktionaalisen riippuvuuden X -> Y, jos kaikille relaation r monikoille t ja t', joille on voimassa t[X] = t'[X], pätee myös t[Y] = t'[Y]. Esimerkiksi {A_1, A_2} -> {A_3} merkitsisi, että attribuuttien A_1 ja A_2 arvot yhdessä määräisivät attribuutin A_3 arvon [Nummenmaa, 1995].

Attribuuttiryhmien r_1 ja r_2 välillä voi olla myös moniarvoinen riippuvuus. Tällöin ryhmän r_1 attribuuttien tietty arvo voi viitata useampaan ryhmän r_2 attribuuttien arvoryhmään. Tämä merkitään r_1 ->> r_2. Eli muodollisemmin, jos relaatiossa r on voimassa moniarvoinen riippuvuus X ->> Y, ja jos on olemassa sellaiset relaation r monikot t ja t', että t[X] = t'=[X], silloin on myös olemassa sellaiset relaation r monikot u, u', että u[X] = t[X], u'[X] = t'[X], u[Y] = t[Y], u'[Y] = t'[Y], u[R\XY] = t'[R\XY] ja u'[R\XY] = t[R\XY], missä R on relaation r tietokantakaavio [Nummenmaa, 1995].

Relaatiolla on yksi tai useampia superavaimia, jotka koostuvat attribuuteista, joiden arvot yksilöivät rivin t (A -> t). Jos ei ole olemassa sellaista superavaimen A aitoa osajoukkoa, joka yksilöisi rivin t, superavain A on myös avain. Avain on siis superavaimen osajoukko.

2.2 SQL-malli

Olkoon T tietokantataulu. T koostuu attribuuteista An; tästä käytetään merkintää: T(A1, A2...An). Jokaisella taululla ja attribuutilla on yksilöivä nimi (otsikko). Tauluun T liittyy myös rivejä, joissa kussakin on n tietoalkiota, yksi jokaista attribuuttia kohden. Tyhjää tietoalkiota merkitään "null". Taulun sisältö voidaan kuvata taulukkona, jossa attribuuttien nimet ovat sarakkeiden otsikoita ja taulun sisältö on riveinä. Tietokanta puolestaan koostuu yhdestä tai useammasta tietokantataulusta.

Kullakin attribuutilla on tietotyyppi t, joka rajoittaa attribuutin arvojoukkoa, eli arvoa, jonka rivin tietty sarake voi saada. Arvojoukko voidaan rajoittaa esimerkiksi kokonaisluvuiksi tai tietynpituisiksi merkkijoiksi. Taulun attribuuttien tyypit merkitään seuraavasti: T(A1 t1, A2 t1, A3 t2, A4 t3).

Taululla on pääavain. Se koostuu attribuuteista, joiden arvot yksilöivät rivin (X-> R). Nämä attribuutit kuvataan merkitsemällä # niiden nimen perään. Taulun T(A1# t1, A2# t1, A3 t2, A4 t3) pääavain on A1, A2. Taulun pääavain on relaationmallin superavain, muttei välttämättä relaatiomallin avain, koska pääavain on voitu määritellä siten, että sen osajoukkokin olisi yksilöivä ominaisuus. Taululla voi olla myös muita yksilöiviä attribuuttiryhmiä (avaimia, uniikkeja ryhmiä), mutta yksi niistä valitaan pääavaimeksi. Muut avaimet kuvataan U(attribuutti1, attribuutti2... attribuuttiN).

Toinen taulun kaltainen rakenne tietokannassa on näkymä. Näkymän kaikki attribuutit viittaavat jonkin taulun johonkin attribuuttiin, näkymät eivät siis sisällä omaa tietoa, vaan niiden sisältö heataan taulusta. Näkymä kuvataan seuraavasti: V*(A3 [R (A1)], A7 [R(A2)]), missä hakasuluissa määritellään taulu, josta attribuutti haetaan ja attribuutin nimi siinä.

Toinen SQL-mallin tukema attribuuttiriippuvuussuhde on vierasavain. Vierasavain tarkoittaa kahden samankokoisen, eri tauluihin kuuluvan, attribuuttiryhmän välistä riippuvuutta, missä toinen attribuuttiryhmä yksilöi toisen. Ainakin toinen ryhmistä on avain omassa taulussaan ja rivit, joissa attribuutit ovat saaneet samat arvot, liittyvät toisiinsa. Taulujen T1(A1#, A2#, A3) ja T2(A1#, A4, A5) välillä siis on vierasavainsuhde liittyen attribuuttiin A1, koska se on molemmissa tauluissa ja taulun T2 avain. Vierasavaimista käytetään merkintää FK(oma_attribuutti1, oma_attribuutti2... oma_attribuuttiN) viitetaulu(vierasattribuutti1, vierasattribuutti2... vierasattribuuttiN): R1(A1#, A2#, A3, FK (A1) R2 (A4)).

SQL-malli on siis relaatiomallia rajoitetumpi.

3. Takaisinmallinnuksen ongelmat ja vaihejako

Kaikki tietokantajärjestelmät eivät myöskään näytä suoraan erilaisia rajoitteita. Rajoite on nimensä mukaisesti tietokannan sisältöä jollain tavalla rajoittava ehto. Tarkkaa virallista määrittelyä asiasta ei ole, mutta esimerkiksi Elmasrin ja Navathen [2000] mukaan kaikkein yksinkertaisimmat rajoitteet määrittelevät yksittäisen tietokantaattribuutin tyypin (esimerkiksi numero- tai kirjainmerkkiaineistoa) sekä mahdollisesti pituuden. Joskus nämä jätetään huomiotta; ehkä siitä syystä, että ne koskevat vain yksittäisiä attribuutteja ja ovat määriteltynä melkein aina tietokannan kaaviossa. Näin ne siis saadaan yleensä aina helposti selvitettyä, eivätkä ne sen vuoksi ole erityisen kiinnostavia.

Tieto siitä, voiko attribuutti saada tyhjän arvon (null), saattaa olla hankalampi selvitettävä, vaikka sekin koskee vain yhtä attribuuttia. Suurin osa tietokantajärjestelmistä tosin tukee suoraan tätä attribuuttirajoitetta, jolloin tyhjä arvo -rajoitteet saadaan suoraan tietokannan luontilauseista. Mutta siltikään ei voida olla varmoja, että tietokannan alkuperäiset kehittäjät olisivat huomanneet määritellä kaikki attribuutit, jotka eivät tyhjää arvoa voi saada.

Jos tietokannassa on paljon aineistoa, voidaan tyhjiä arvoja sallivia attribuutteja kohtalaisen luotettavasti päätellä tietokannan sisällön perusteella. Yleensähän aineistoa on paljon, jos tietokannan takaisinmallinnus ylipäänsä on tarpeen. Täysin varmaa tällainen päättely ei kuitenkaan ole. Tapaus, jossa attribuutti saa tyhjän arvon, voi olla niin harvinainen, ettei se satu mukaan isoonkaan aineistoon, vaikka periaatteessa mahdollinen olisikin.

3.1 Optimointeja ja rajoitteita

Vaikkei ohjelmakoodista löytyvien rakenteiden kaltaisia optimointeja ei tietokannasta yhtä hankalina löydykään, tiedon tulkintaa vaikeuttavia rakenteita on voitu sisällyttää tietokantaankin. Kannassa voi esimerkiksi olla tietoalkio, jonka sisällön perusteella pitää tulkita jonkin toisen tietoalkion sisältöä (olemassaoloriippuvuus). Tällainen tapaus voisi olla, työtehtävän mukaan laskettavat lisäbonukset: Tietokannan yksi alkio on henkilön peruspalkka, mutta kokonaispalkka lasketaan ohjelmakoodissa työtehtäväkohtaisten lisäbonusten mukaan. Tietokannan dokumentointi onkin erityisen tärkeää, koska tehtävä ohjelma rakentuu tietokannan päälle ja pieni tulkintavirhe tietokantadokumentoinnissa voi aiheuttaa suuria muutoksia ohjelmakoodiin.

Monimutkaisemmat rajoitteet koskevat yleensä useampia attribuutteja ja ovat entistä hankalammin selvitettävissä. Uudemmat relaatiotietokannat tukevat avain (ainutlaatuisuus)- ja pääavainrajoitteita, sekä avainten suhteita toisiinsa, ns. vierasavainsuhteita. Verrattuna esimerkiksi oliomalliin eivät uudemmatkaan SQL-pohjaiset relaatiotietokannat tarjoa kovin monipuolisia työkaluja suhteiden kuvaamiseen.

Erityisesti taulujen välisiä suhteita ei relaatiomallissa edes voi kuvata kunnolla.

Vierasavaimet kertovat jonkinlaisesta suhteesta, mutta nekään eivät tarkenna, millaisesta suhteesta on kyse. Rajoitteita voi päätellä analysoimalla ohjelmallisesti tietokannan sisältöä ja kaaviota sekä hankkimalla tietoja tietokannan kohdealueesta.

3.2 Häiriö ja piilotieto

Takaisinmallinnusmenetelmien tuloksia voidaan arvioida niiden sisältämän häiriön (noise) ja piilotiedon (silence) avulla. Näistä häiriö on yksinkertaisempi käsite.

Häiriötä syntyy Henrardin ja Hainautin [2001] mukaan, kun menetelmä löytää tietokannasta jotain, mitä siellä ei ole. Tietokantaan voidaan vahingossa syöttää väärää tietoa (näppäilyvirheet, väärin luettu tai kuultu tieto). Myös huono tietokantasuunnittelu ja siitä aiheutuvat väärinkäsitykset lisäävät väärän tiedon esiintymistä.

Rakenteiden optimointi aiheuttaa myös häiriötä, koska kannasta voi saada väärän käsityksen liiallisten optimointien jälkeen. Ohjelmakoodia tutkittaessa häiriötekijöitä ovat koodin vanhentuneet osat ja sellaiset kohdat, joita ei enää käytetä. Jos niitä kuitenkin tulkitaan takaisinmallinnettaessa, saattavat ne heijastaa vanhentunutta tietoa tietokannan kuvaukseen.

Häiriön haittavaikutuksena olemattomien asioiden poissulkemisessa kuluu aikaa, mikä on siis haitan hinta. Toisaalta piilotieto jää nimensä mukaisesti takaisinmallinnus-menetelmän ulottumattomiin. Tässä tapauksessa haitan laskeminen on hankalampaa.

Jos piiloon jäävä tieto selviää myöhemmin, voidaan sen haitta yksinkertaisesti laskea työmääräksi, joka vaaditaan tiedon liittämiseksi kehitettyyn tietomalliin. Tämä työmäärä jo sinällään on tietysti usein suurempi kuin häiriön aiheuttama, koska jonkin olemassaolemattomuuden tarkistaminen on monesti helpompaa kuin alkaa myöhemmin muokkaamaan valmista mallia.

Toisaalta piilotieto ei ehkä selviä ollenkaan, jolloin se aiheuttaa myöhemmin ongelmia järjestelmän toimiessa epätarkasti tai jättäessä jotain huomiotta. Tällaiset ohjelmavirheet voivat aiheuttaa pitkällä aikavälillä tietojärjestelmää käyttävälle taholle suuriakin tappioita. Toki myös häiriö saattaa jäädä huomaamatta, jolloin se voi olla yhtä haitallista kuin piilotieto.

Huono tietokantasuunnittelu saattaa aiheuttaa yhtä lailla piilotietoa kuin häiriötäkin. Yhdistettynä puutteelliseen tai puuttuvaan dokumentointiin voi osa tietokannan rakenteiden merkityksistä jäädä pimentoon. Myös ohjelmakoodissa saattaa olla tällaisia epäselviä optimointeja.

3.3 Inhimillisen päätöksenteon tarve

Tietokannan takaisinmallintaminen voi olla huomattavan suuri työ. Näin on erityisesti, jos mallinnetaan satoja tauluja sisältäviä tietokantoja. Automatisointi on siis tarpeen. Ongelmana on, että jokainen tietokanta on erilainen, mitään kattavia yleispäteviä säännöstöjä on hankalaa luoda ja vielä vaikeampaa on kehittää ohjelmaa, joka pystyisi itsenäisesti mallintamaan oikein.

Automatisointi ei kuitenkaan ole yksiselitteisen mahdotonta; osa prosesseista

voidaan automatisoida täysin. Esimerkkinä tällaisesta tilanteesta on vierasavain-suhteiden tunnistamisen tiettyjen sääntöjen perusteella (vierasavaimen toinen puoli on pääavain ja toisella puolella on samannimiset ja -tyyppiset attribuutit) [Henrard et al., 1999]. Osittain voidaan automatisoida esimerkiksi attribuuttiyhdisteiden havaitseminen. Kolmanneksi voidaan rakentaa työkaluja, jotka osaavat automaattisesti luoda mallintajan avuksi erilaisia raportteja tietokannasta.

3.4 Takaisinmallinnuksen tiedonlähteet

Tietokannan merkityksiä voidaan kannan kaavion lisäksi selvittää tietokannan sisällön ja sitä käyttävän ohjelmakoodin perusteella. Näistä kaaviontulkitseminen on kaikkein nopein ja varmin tapa, ja tietokannan sisällönkin käyttäminen on koodin tulkitsemista helpompaa. Henrard et al. [1999] tukevat kaaviontulkitsemista ensimmäisenä työvaiheena neljästä syystä.

Ensinnäkin välimatka käsitteellisen mallin ja ohjelmiston käytännön toteutuksen välillä on tietorakenteissa pienempi kuin toiminnallisissa osuuksissa. Toiseksi tietorakenteet ovat yleisesti ottaen sovelluksen vakain, hitaimmin muuttuva, osa. Kolmanneksi tietorakenteet ovat toiminnoista erillään jopa hyvin vanhoissa sovelluksissa. Neljänneksi ohjelmakoodin tulkitseminen on paljon helpompaa tietorakenteiden selvittämisen jälkeen.

Kaikkea tietoa ei kuitenkaan kaaviosta löydy. Koska relaatiomalli ei tarjoa suurta ilmaisuvoimaa esimerkiksi eri olioiden (taulujen) suhteista, tarvitaan avuksi ohjelmakoodia. Tieto siitä, miten tauluja tulisi käyttää, ja miten ne toimivat yhteen löytyy usein upotettuna ohjelman käyttämiin SQL-lauseisiin.

Tämä koodin tulkitseminen on hidasta, mutta usein ainoa tapa selvittää rakenteiden merkityksiä, jos dokumentointi on kehnoa tai puutteellista. Varsinaisten upotettujen SQL-lauseiden analysoimisen lisäksi voidaan tutkia sitä, mitä ohjelmakoodi tekee tietokannasta haetuille lauseille ja kuinka tietokantaan tallennettavaa tietoa käsitellään. Tällainen tulkinta on kuitenkin hankalaa ja aikaavievää, erityisesti jos ohjelmaa ei ole rakennettu hyvien suunnitteluperiaatteiden mukaisesti.

Muiksi takaisinmallinnuksen tiedonlähteiksi Henrard et al. [1999] listaavat ohjelman tulosteet, tietokantajärjestelmän ominaisuudet, olemassaolevan dokumentaation (tämä on tietenkin kaikkein paras tiedonlähde, jos se on ajan tasalla), käyttäjien ja ohjelmistonkehittäjien haastattelut sekä mahdolliset tiedot käyttöympäristöstä. Blahan [1997] syötetietolistauksessa otetaan erityisesti huomioon, että käyttöalueesta voi olla olemassa käsitehakemisto. Tässä tutkielmassa keskityn tietokantalähtöisiin tiedonlähteisiin, eli tietokannan kuvaukseen ja sisältöön sekä jonkin verran myös tietokantaa käyttävän ohjelman ohjelmakoodiin.

3.5 Takaisinmallinnuksen vaiheet

Henrard et al. [2002] huomauttavat, että yleensä harkitaan kahta erilaista järjestelmänmuuntostrategiaa ja sitä myötä myös kahta erilaista takaisinmallinnusstrategiaa: joko muutetaan koko järjestelmä kerralla tai pieniä askeleita ottaen.

Jahnke et al. [1997] jakavat tietokannan takaisinmallinnuksen kolmeen osan: tietokantakaavion takaisinmallinnus, tietokannan sisällön takaisinmallinnus ja tietokantaa käyttävien ohjelmien takaisinmallinnus. Henrard et al. [2002] tekevät samanlaisen jaon.

Ensimmäisen vaiheen Henrard et al. [1999] jakavat kahteen osaan: tietorakenteen selvittäminen ja tietorakenteen käsitteellistäminen. Myöhemmin Henrard ja Hainaut [2001] ottivat mukaan valmisteluvaiheen edeltämään näitä kahta varsinaiseen mallintamiseen kuuluvaa osaa. Valmisteluvaiheessa kerätään ja arvioidaan tarvittavia tietolähteitä ja tutustutetaan mallintaja kohdealueeseen haastattelujen ja muiden esitelmien avulla. Jälkimmäiset osat keskittyvät kahden erilaisen kaavion kehittämiseen.

Tietorakennetta selvitettäessä kootaan looginen kaavio, johon kuuluvat kaikenlaiset rakenteet ja rajoitteet. Henrard et al. [1999] jakavat tämän vaiheen neljään alivaiheeseen. Ensimmäiseksi analysoidaan tiedonkuvauskieli (Data Description Language). Jäsentämällä kuvaus saadaan aikaan raaka, fyysinen kaavio. Toisessa alivaiheessa yhdistetään fyysiset kaaviot, jos niitä on ensimmäisessä vaiheessa tehty useampia kuin yksi. Kolmanneksi, kaavion jalostusvaiheessa, etsitään piiloon tai kokonaan toteuttamatta jääneitä rakenteita. Ensin kehitetään hypoteeseja tietokannan piilorajoitteista, sitten käytetään muita tiedonlähteitä varmistamaan hypoteesirajoitteiden oikeellisuus. Hypoteesi joko muokataan, hylätään tai hyväksytään, minkä jälkeen se liitetään kaavioon. Neljänneksi poistetaan kaaviosta kaikki tekniset optimoinnit, kuten indeksit ja ryväsrakenteet. Tämän jälkeen tietorakenteen selvittäminen päättyy, eikä tuloskaavio ole enää tietokantajärjestelmän kanssa yhteensopiva kahdesta syystä [Henrard et al., 1999]: Ensinnäkin kaavio saattaa sisältää useamman kuin yhden tietokannan ja nämä tietokannat voivat olla useammasta erilaisesta järjestelmästä. Toiseksi mukaan on otettu rakenteita, joita tietokantajärjestelmä ei luultavasti tue.

Kolmas päävaihe on siis tietorakenteen käsitteellistäminen. Tämä tarkoittaa eikäsitteellisten rakenteiden muokkaamista tai poistamista, toistamisen vähentämistä ja tietokantajärjestelmälle ominaisten sekä muiden teknisten optimointien tulkitsemista. Lopulta myös tietokantajärjestelmän rakenteet korvataan valitulla käsitteellisellä mallilla.

4. Tietokantojen suunnittelun yleisemmät käytännöt

4.1 Tietokantojen suunnittelu

Halutessa selvittää tietokannan merkityksiä voidaan aloittaa takaisinmallintaminen tietokannan alkuperäisestä mallinnuksesta. On siis pohdittava, miten tietokanta luotiin, ja millaisia kohteita yleensä mallinnetaan tietyiksi tietokannan rakenteiksi.

Premerlani ja Blaha [1994] listaavat takaisinmallinnuksen syylajeiksi paradigman vaihtamisen (siirtyminen relaatiotietokannasta oliopohjaiseen tietokantaan), paradigman esiintymän vaihtamisen (siirtyminen relaatiotietokantajärjestelmästä toiseen) ja yleisen halun selvittää tietokannan tietojen merkityksiä.

He huomauttavat myös, että käytännössä tietokantojen suunnittelijat monesti rikkovat hyvän suunnittelun periaatteita. Näin ollen tarvitaan myös sellaisia takaisinmallinnustekniikoita, jotka eivät vaadi mallinnettavan tietokannan olevan esimerkiksi kolmannessa normaalimuodossa.

Hyvän suunnittelun periaatteiden rikkomisen lisäksi kantakaavioissa saattaa olla varsinaisia virheitäkin [Blaha, 1997].

4.2 Luokat ja taulut

Usein luokat on mallinnettu tietokantaan tauluiksi ja luokkien attribuutit tietokannan sarakkeiksi [Premerlani and Blaha, 1992]. Luokkia voidaan myös yhdistää tai jakaa useammaksi tauluksi. Toisaalta myös yksi-yhteen -suhteet voidaan toisinaan nekin mallintaa tauluksi. Attribuutittomat luokat taas voidaan jättää kokonaan mallintamatta.

Luokkien välisiä suhteita voidaan relaatiomallissa kuvata vierasavaimilla [Premerlani and Blaha, 1992]. Jos vierasavain viittaa yliluokkaan, sitä on voitu työntää alaspäin luokan kaikkiin aliluokkiin. Näin on voitu saada aikaan samaan tauluun useita vierasavaimia tai useaan tauluun viittaava vierasavain. Jälkimmäiset usein muunnetaan tauluiksi. Suunnittelija voi muuttaa tauluiksi myös ei-n*n-suhteita, jos esimerkiksi suhteella itsellään on attribuutteja.

Luokkien yleistyksien ja erikoistuksien mallintamiseen on käytössä useita eri tapoja [Premerlani and Blaha, 1992]. Selkein tapa on luoda oma taulu yliluokalle ja sen kaikille aliluokille, missä yliluokalla on sarake, joka määrittää, mihin aliluokaan kukin rivi kuuluu. Toinen tapa on luoda yksi taulu sisällyttämällä eri aliluokkien attribuutit yliluokkaan, jolloin suurikin osa rivin arvoista voi olla tyhjiä. Kolmanneksi voidaan luoda aliluokista omat taulut, joissa on kaikissa toistettuna yliluokan attribuutit.

4.3 Identiteetti

Tietokantatiedon (taulun) identiteetti on yksi selkeimmistä ja tärkeimmistä mallinnusongelmista: millaiset attribuutit määrittävät jokaisen rivin kussakin taulussa erilaiseksi. Tärkeä se on muun muassa sen vuoksi, että suuri osa muista takaisinmallinnusongelmista pohjautuu identiteettiin (esimerkiksi vierasavaimet ja taulun merkitys ylemisemmin).

Ratkaisutapoja ovat Blahan [1997] mukaan keinotekoinen identiteetti, arvoperustainen identiteetti, yhdisteidentiteetti ja peritty identiteetti. Keinotekoinen identiteetti on vain identifiointitarkoitusta varten tauluun luotu attribuutti, jonka tunnistaa yleensä siitä, että sen nimi päättyy "id". Toki myös muiden attribuuttien nimi voi päättyä samoin.

Arvoperustainen identiteetti tarkoittaa, että identifioivaksi attribuutiksi on valittu jokin tai joitain luonnollisia attribuutteja, jotka määrittävät tietokantataulun rivin. Yhdisteidentiteetti puolestaan on kahden edellisen yhdistelmä, esimerkiksi pankkitilin voi määrittää tilinumero ja pankkitaulun keinotekoinen identiteettiattribuutti [Blaha, 1997].

Perityssä identiteetissä taulun identiteetti on viittaus toiseen tauluun [Blaha, 1997]. Näin tapahtuu, kun taulujen välillä on vierasavainsuhde, ja vierasavaimen attribuutit ovat molempien taulujen identiteettiattribuutit.

4.4 Tyyliseikat

Tietokannan rakennustyylikin kuuluu takaisinmallinnusta tehdessä huomioonotettaviin ulottuvuuksiin. Tyyliratkaisuita ovat Blahan [1997] mukaan yhdenmukaisuus, mallipohjaisuus, mallinnusparadigma, tietotyyppiyhdenmukaisuus, kaavionluontitapa, kaavion ikä, kaaviota käyttävän ohjelman ikä, optimoinnin aste ja keinotekoiset rakenteet.

Yhdenmukaisuus tarkoittaa, että mallintaessa ratkaistaan samanlaiset ongelmat samalla tavalla. Mallipohjaisuus puolestaan kertoo, onko tietokanta luotu ensin huolellisesti rakennetun mallin pohjalle vai onko se kasattu nopeasti ja paikattu myöhemmin toimivaksi.

Mallinnusparadigmoja ovat muun muassa UML ja ER. Tietotyyppiyhdenmukaisuus taas kertoo, onko kaaviossa käytetty samanlaisia tietotyyppejä samanlaisiin kohteisiin. Kaavionluontitapoja ovat automaattinen generointi ja käsinkirjoittaminen. Kaavion ja kaaviota käyttävän ohjelman iät kertovat osaltaan siitä, millaista tyyliä kaaviosta kannattaa odottaa löytävänsä.

Optimoinnin aste vaihtelee iän ja suorituskykyodotusten mukaan. Myös ohjelmistokehittäjillä on omia mielipiteitä siitä, pitäisikö suosia tehokkaita vai uudelleenkäytettäviä ja helpostiymmärrettäviä rakenteita. Keinotekoisia rakenteita taas ovat esimerkiksi väliaikaistiedot, jotka on jostain syystä tallennettu tietokantaan.

4.5 Tietokannan koko ja metadata

Premerlani ja Blaha [1994] listaavat myös muita tietokantojen takaisinmallinnus-kokemuksista peräisin olevia periaatteita. Ensinnäkin he ovat huomanneet, että vaikka isoissa tietokannoissa taulujen suuri määrä hankaloittaa kaikkien niiden läpikäymistä, saa suuri koko myös suunnittelijat käyttämään keskenään samanlaisia rakenteita.

Toisaalta se, että kahden attribuutin pituus eroaa toisistaan, ei tarkoita, että ne viittaisivat eri asiaan (estäen vierasavainsuhteet). Pituuserot voi selittää esimerkiksi

sillä, että toisessa taulussa attribuutin perään on lisätty jotain vakiotäytettä.

Erityisen hankalaa on tekijöiden mukaan huomata tietokantaan itseensä piilotettu metadata. Tällaista on esimerkiksi sarakkeiden nimet, jonkin taulun rivien arvoina. Samoin hankalaa on huomata, jos yksi attribuutti sisältää useampaa tietoa (komposiitti).

5. Algoritmisia ratkaisuja ja keinoja riippuvuuksien löytämiseksi

Tietokantojen attribuuttien, sekä samassa taulussa olevien että eri taulujen attribuuttien, välillä voi olla erilaisia riippuvuuksia. Näitä ovat vierasavainsuhteet, toistoattribuutit ja olemassaoloriippuvuudet. Toistoattribuutti tarkoittaa attribuuttia, jonka tieto voidaan johtaa muista attribuuteista. Olemassaoloriippuvuus puolestaan tarkoittaa, että attribuutin arvo riippuu jonkin toisen attribuutin arvosta. Luvun ensimmäisessä kohdassa käsittelen Flachin ja Savnikin koneoppimisnäkökulmaa, toisessa taas Limin ja Harrisonin kehittämää algoritmia riippuvuuksien löytämiseksi. Kolmanneksi on vuorossa Soutoun SQL-kyselyitä käyttävä menetelmä. Lopuksi neljäs kohta esittelee Henrardin ja Hainautin listaamia menetelmiä lähdekoodin käyttämiseksi.

5.1 Koneoppimisen näkökulma

Flach ja Savnik [1999] ovat tarkastelleet erilaisia rajoitustenetsintäalgoritmeja logiikkaohjelmointikielten ja koneoppimisen näkökulmasta. Logiikkaohjelmoinnin sanastolla ilmaistuna sääntöjä (riippuvuudet) etsitään faktoista (tietokantatauluista). Flach ja Savnik kokeilivat kaikkiaan kolmea induktioperustaista (tietokannan sisällöstä sääntöjä päättelevää) menetelmää.

Ensimmäinen kokeiltu algoritmi oli yksinkertainen, yleisestä-erityiseen (top-down)-pohjainen: käydään kaikki attribuuttiparit läpi tietokannan sisältöä tutkien ja tarkistetaan, löytyykö riippuvuutta. Ellei löydy, riippumatonta osuutta kasvatetaan. Tätä jatketaan, kunnes havaitaan riippuvuus tai huomataan, ettei riippuvuuksia ole.

Toisessa menetelmässä, jota Flach ja Savnik kutsuvat kaksisuuntaiseksi induktioalgoritmiksi, käytetään sekä yleisestä-erityiseen että erityisestä-yleiseen (bottom-up) -lähestymistapoja. Ensin sovelletaan yleisestä-erityiseen -menetelmää keräämään lista niistä riippuvuuksista, jotka aineisto todistaa vääräksi. Nämä riippuvuudet muodostavat negatiivisen peitteen (negative cover). Sen jälkeen käytetään ensimmäisen menetelmän yleisestä-erityiseen -tapaa riippuvuuksien etsimiseksi, mutta tällä kertaa tutkitaan vain niitä riippuvuuksia, joita negatiivisesta peitteestä ei löydy.

Kolmas algoritmi on puhdas erityisestä-yleiseen -menetelmä. Siinä luodaan negatiivinen peite, kuten edellä. Tässä tapauksessa se vain käydään läpi kerran tutkien kaikkein harvemmin rikottuja riippuvuuksia.

Erityisesti ensimmäisen menetelmän vaatima prosessointi kasvaa tietenkin suuresti attribuuttien määrän kasvaessa. Flach ja Savnik totesivatkin menetelmän liian hitaaksi. Toinen algoritmi on jonkin verran parempi, mutta kolmas on selkeästi tehokkain. Flach ja Savnik eivät kuitenkaan huomioineet Limin ja Harrisonin [1997] pari vuotta aikaisempaa tutkimusta, jossa he olivat kehittäneet optimointeja yleisestä-erityiseen -mallille.

5.2 Limin-Harrisonin –algoritmi

Lim ja Harrison [1997] ovat jalostaneet relaatiomallin pohjalta olemassaolevia

algoritmeja saadakseen aikaan tehokkaan algoritmin toiminnallisten riippuvuuksien löytämiseksi tietokannasta. Lisäksi he ovat kehittäneet tehokasta tapaa löytää virheet tietokannan sisällöstä.

Aiemmat tällaiset algoritmit ovat yrittäneet luoda pienen määrän hypoteeseja riippuvuuksista ja tarkistaneet niitä sitten tietokantaa vastaan; näiden menetelmien ongelmana on tehottomuus suurissa tietokannoissa. Menetelmät myös olettavat tietokannan tiedon olevan virheetöntä, mikä on tosielämässä harvinainen tilanne mittausvirheiden ja tiedon vääränlaisen koodauksen vuoksi.

Tekijät jakavat algoritmit kahteen ryhmään: yleisestä-erityiseen- ja erityisestä-yleiseen -algoritmeihin. Ensimmäisessä etsitään ensin mahdollisia riippuvuuksia ja sitten tarkistetaan niiden paikkansapitäminen tietokannan sisällön perusteella.

Eksponentiaalisesti kasvavaa (Flachin ja Savnikin mukaan yleisestä-erityiseen -menetelmän tehottomaksi tekevää) sarakeyhdistelmien määrää kaventamaan on kehitetty muutamia yksinkertaistuksia. Jos pienemmän avaimen on jo huomattu aiheuttavan riippuvuuden, ei tarvitse tarkistaa suurempia superavaimia, jotka sisältäisivät kaikki avaimen attribuutit. Mahdollisen attribuuttiyhdistelmän kokoa voidaan myös pienentää tai hankkia lisätietoa tietokannan kohdealueesta. Hypoteesien vahvistaminen voidaan tehdä pareissa, mutta tehokkaampi tarkistus saadaan jos taulu ensin järjestetään riippumattomien muuttujien mukaan.

Erityisestä-yleiseen -menetelmät puolestaan tutkivat Limin ja Harrisonin määrittelemän mukaan pelkkää tietokannan sisältöä ottaen kohteeksi kaksi attribuuttiryhmää kerralla ja tutkien, aiheuttavatko tietyt arvot toisessa ryhmässä tiettyjä arvoja toisessa ryhmässä. Tällainen on tietenkin laskennallisesti erittäin raskasta.

5.3 Moniasteisten suhteiden etsiminen

Soutou [1996] on kehittänyt relaatiomallia ja SQL-mallia käyttäen menetelmän moniarvoisten suhteiden löytämiseksi käytössä olevasta tietokannasta. Menetelmän olettaa lähtövaatimuksenaan, että avainattribuuttien nimissä ei esiinny homonymiaa, muttei vaadi rajoitteita etukäteen tiedetyksi.

Soutoun menetelmä käyttää aineistona otosta tietokannan sisällöstä ja uuttaa niistä havaitsemilleen suhteille asteet. Tämän menetelmän ongelmana on, että tietokannan suunnittelija on voinut sallia N:M-suhteet, mutta senhetkinen tietokannan sisältö noudattaa rajoitetumpaa suhdetta (N:1 tai 1:1 –suhdetta).

Menetelmän ensimmäinen vaihe on tietokantataulujen kuvausten noutaminen. Toisessa vaiheessa suoritetaan varsinainen suhdeasteiden tunnistaminen ja kolmannessa vaiheessa annetaan tietokantakuvauksen perusteella nimet suhteeseen osallistuville attribuuteille.

Näistä toinen vaihe on takaisinmallinnusnäkökulmasta kiinnostavin. Monista muista menetelmistä poiketen Soutoun menetelmä käyttää tietokantaan suoritettavia SQL-alikyselyitä.

Käytännössä menetelmä käyttää kahdentyyppisiä alikyselyitä: positiivisia ”on

olemassa” -kyselyitä ja negatiivisia ”ei ole olemassa” -kyselyitä. Positiiviset kyselyt tarkistavat onko taulussa olemassa sellaista riviä, jossa tietyn vierasavaimeen liittyvän attribuutin arvon vaihtelisi; negatiiviset kyselyt toimivat tietenkin päinvastoin. Positiivisia kyselyitä tarvitaan kutakin taulua kohtaan niin monta kuin taulussa on pääavaimeen kuulumattomia vierasavainattribuutteja.

Negatiivisia kyselyissä puolestaan tarkistetaan, ettei ole olemassa kahta riviä, joissa pääavainattribuutit olisivat samoja. Tämän vuoksi niitä tarvitaankin saman verran kuin on pääavainattribuutteja.

Soutou on havainnut myös ongelmia menetelmässään. Ensinnäkin positiiviset alikyselyt ja negatiiviset alikyselyt voivat olla ristiriidassa esiintyessään samassa hakulauseessa. Tällöin mitään tuloksia ei tietenkään voida saada, mikä voi johtaa piilotietoon. Tältä voidaan Soutoun mukaan välttyä huomioimalla alikyselyiden asteet ja huolehtimalla siitä, ettei samaan attribuuttiin liittyen ole alikyselyitä yhdessä kyselyssä.

Toiseksi on mahdollista, että samassa kyselyssä on kaksi samaa asiaa kokeilevaa positiivista ja negatiivista alikyselyä. Ongelman ratkaisuksi Soutou ehdottaa kyselypuiden muodostamista.

5.4 Menetelmiä ohjelmakoodin tulkitsemiseen

Henrard ja Hainaut [2001] ovat keränneet menetelmiä ohjelmakoodin tulkitsemiseksi, koska heidän mukaansa rajoitukset useimmiten määritellään juuri ohjelmakoodissa. Nämä menetelmät tutkivat, esiintyykö kahden tai useamman attribuutin välisiä tarkistuksia. Henrard ja Hainaut pohjasivat tutkimuksensa SQL-malliin.

Ensimmäinen menetelmä on muuttujariippuvuuskaavio (variable dependency graph), joka on helposti laskettava versio tietovirtakaaviosta. Kaaviossa jokainen muuttuja esitetään solmuna, nuoli solmujen välille piirretään suorasta yhteydestä (sijoitus, vertailu tms.) Jos muuttujien välillä on translatiivinen suhde, ne voivat esittää samoja olioita, jakaa samat arvot tai jotain muuta.

Henrard ja Hainaut havaitsivat, että muuttujariippuvuuskaaviota tulkitsemalla, voidaan päätyä kolmen eri lajin piilotietoon ja kahden eri lajin häiriöön. Ensinnäkin, jos otetaan huomioon vain sijoitussuhteet, saattaa riippuvuuksia jäädä huomaamatta. Tästä ongelmasta selviää, ottamalla huomioon monipuolisesti erilaisia vuorovaikutuksia muuttujien välillä (esimerkiksi vertailut). Toinen mahdollinen piilotieto johtuu siitä, että ohjelmointikielen rakenne on liian epäselvä, jotta tulkintaa voitaisi tehdä.

Kolmas piilotiedon syntymisvaihtoehto on ohjausrakenteiden huomiottajättäminen. Ohjelmakoodissa voi olla määriteltynä, että muuttujan a ollessa arvoltaan jokin tietty, muuttujan b arvoa muutetaan. Tällainen riippuvuus jää huomaamatta käytäessä läpi pelkkiä yksittäisiä lauseita.

Häiriötä puolestaan voi syntyä, jos tiettyä muuttujaa käytetään useammassa erillisessä prosessissa vertailemaan useampia erillisiä tietokantamuuttujia [Hehrard and Hainaut, 2001]. Tällöin muuttujakaavio voi alkaa seuraamaan myös tiettyyn muuttujaan

vaikuttamattomia muuttujia. Toiseksi johonkin tietokannasta aluksi poimittuun muuttujaan on voitu myöhemmin sijoittaa jokin täysin toinen arvo. Tämän arvon sijoittaminen jonkin toisen attribuutin arvoksi ei luo riippuvuutta näiden attribuuttien välille.

Ohjelman siivutuksessa ohjelmakoodista poimitaan ohjelmasta kaikki ne kohdat, jotka voivat tietyssä kohdassa vaikuttaa tietyn muuttujan arvoon. Henrard ja Hainaut huomauttavat, että Horwitz et al. kehittivät tästä eteenpäin järjestelmäriippuvuus-kaavion, joka ottaa huomioon myös toimenpidesarjat. Henrard ja Hainaut kehittävät kaaviota vielä eteenpäin lisäämällä siihen satunnaisen ohjausvirran. Kaaviossa eri koodirivit ja sijoitukset ovat solmuja, joiden välillä kulkee siis ohjaus- ja vaikutusnuolia.

Ohjelman viipalointi tehdään kulkemalla kaaviossa kahdessa vaiheessa [Hehrard and Hainaut, 2001]. Ensin tutkitaan, mitkä ohjelman sisäiset solmut tavoittavat solmun s, toisessa vaiheessa tutkitaan, mitkä ohjelman kutsumat toiminnot ja ohjelmaa kutsuvien toimintojen kutsuvat toiminnot tavoittavat solmun s.

Ohjelman siivuttamista voidaan käyttää kolmella eri kyselytavalla [Hehrard and Hainaut, 2001]. Yksinkertaisessa ohjelmasiivutuksessa tutkitaan, edeltääkö jonkin attribuutin tietokantaan kirjoittamista jonkin toisen attribuutin lukeminen. Tällöinhän kyseessä voisi olla riippuvuus (tarkistetaan sopiiko luettavan attribuutin arvo kirjoitettavaan). Näin ei kuitenkaan aina ole, jos attribuutin arvot esimerkiksi syystä tai toisesta pitää lukea ennen kirjoittamista, mutta mitään vertailua ei suoriteta. Menetelmä voi siis aiheuttaa huomattavan paljon häiriötä.

Tietovirtaohjelmasiivutus (dataflow program slicing) keskittyy riippuvuutta kuvaaviin nuoliin ja jättää ohjausvirtaa kuvaavat nuolet huomiotta [Hehrard and Hainaut, 2001]. Tämän vuoksi tekniikka ei kerro kaikkia riippuvuuksia (jättää piilotietoa), muttei myöskään tuota häiriötä.

Tietovirtaohjelmasiivutekniikkaa voi parantaa seuraamalla muuttujia [Hehrard and Hainaut, 2001]. Kun viipaleessa on luku- ja kirjoitusohjeita, voidaan tutkia muuttujan matkaa ohjelmassa ja analysoida siihen liittyviä ohjelmakoodirivejä.

6. Takaisinmallinnusmenetelmien luetteloinnit

Koko takaisinmallinnusprosessin kulkua määritteleviä käytäntöjä on luokiteltu eri tavoin. Aliluvuissa käsittelen kaksi jakoperustetta.

6.1 Pedro-de-Jesuksen ja Sousan jakoperusteet

Pedro-de-Jesus ja Sousa [1999] jakavat eri menetelmät ryhmiin viiden tekijän perusteella. Nämä tekijät ovat vaaditun tiedon muoto, vaaditusta tiedosta tehtävät oletukset, menetelmän tuottama tieto, menetelmän käytännöt ja mitä uutta menetelmät ovat tuoneet.

Jako toimii mielestäni hyvin mallinnusmenetelmiä opiskelevan tarpeisiin. Toisaalta olisi hyödyllistä jaotella menetelmiä myös keskittyen tarkemmin sopivaa menetelmää etsivän tarpeisiin. Tästä näkökulmasta ei oikeastaan ole olennaista, mitä uutta menetelmässä on ollut, tai edes mitkä ovat menetelmän käytännöt, koska menetelmän käytännön toteutus tuskin muodostaa esteitä, joita saanti- ja ulostulotietojen ja oletusten määrääminen ei jo olisi kattanut.

Pedro-de-Jesus ja Sousa määrittelevät myös kahdeksan ominaisuutta lajittelemaan takaisinmallinnustekniikoita. Näistä kolme ensimmäistä kertovat menetelmän vaatimista esitiedoista ja syötelajeista.

*Semanttinen tieto*määrittelee, kuinka paljon mallintaja tietää järjestelmästä. Tällä ominaisuudella on kolme mahdollista arvoa. Joko mallintaja tuntee attribuuttien merkityksen tai ainakin pystyy ratkomaan ristiriitaisuudet tai sitten mallintaja ei tunne järjestelmää ollenkaan. Nämä ristiriitaisuudet voivat johtua homonyymeistä (attribuuteista, joiden nimi on sama, mutta jotka viittaavat eri asioihin) tai synonyymeistä (attribuuteista, joiden nimi on eri, mutta jotka viittaavat samaan asiaan). Toinen ominaisuus liittyy *tietokannan sisältöön*: käyttääkö menetelmä sitä vai ei. Kolmas ominaisuus kertoo *ohjelmakoodin hyödyntämisestä*: koodia, joko käytetään tai ei, ja jos koodia käytetään, menetelmä joko olettaa, että siinä voi olla virheitä, tai että koodi on virheetöntä. Neljä seuraavaa ominaisuutta määrittävät, minkälaisia oletuksia tietokannan rakenteesta tehdään. *Avainriippuvuusominaisuus* kertoo, olettaako menetelmä, että taulujen avaimet ja erityisesti pääavaimet ovat tiedossa. *Vierasavainriippuvaisuusominaisuus* taas kertoo, tiedetäänkö vierasavainsuhteita. *Ei-avainperustaiset toiminnalliset riippuvaisuudet tai kolmas normaalimuoto* –ominaisuus tarkoittaa, että tietokannan tarvitsee olla kolmannessa normaalimuodossa tai avaimiin perustumattomat riippuvuudet ovat tiedossa. Toisaalta *ei-avainpohjaiset sisältymisriippuvuudet* –ominaisuus kertoo, oletetaanko vierasavaimiin liittymättömät sisältymisriippuvuudet tiedetyiksi. Tämä siis tarkoittaa esimerkiksi yli- ja aliluokan suhdetta. Viimeisenä ominaisuutena määritellään *käyttäjän vuorovaikutuksen määrä*, eli millaisissa tilanteissa käyttäjän vuorovaikusta tarvitaan.

Luonnollisesti tarkkaa tietoa attribuuttien merkityksistä vaativa järjestelmä ei tarvitse niin paljon muuta tietoa mallinnuskohteesta kuin vähemmän tietoa vaativa.

Toisaalta tiukat vaatimukset riippuvuusominaisuuksissa heijastuvat vähempänä käyttäjävuorovaikutuksen tarpeena.

6.2 Henrard-Hick-Thiran-Hainaut –jako

Henrard et al. [2002] kuvailevat kuutta erilaista tapaa suorittaa takaisinmallinnus. Näiden tapojen avulla voidaan myös jakaa takaisinmallinnusmenetelmät kuuteen ryhmään.

Jako ryhmiin tapahtuu kahden ulottuvuuden perusteella. Tietokantaulottuvuus määrittää, miten tarkasti tietoa käsitellään. Mahdollisuutena on jäädä fyysiselle tasolle, jolloin entiset rakenteet mallinnetaan mahdollisimman läheisiksi uuden järjestelmän rakenteiksi. Tällöin ei siis yritetä millään lailla ymmärtää tiedon merkityksiä. Toinen vaihtoehto on pyrkiä käsitteelliselle tasolle, eli merkityksien ymmärtämiseen.

Prosessissa avustavia työkaluja on olemassa molempien tasojen menetelmille. Fyysisellä tasolla tarvitaan vain yksinkertaisia työkaluja, kuten määrittelykielen tulkkia, yksinkertaista kaaviomuunninta ja määrittelykieligeneraattoria. Käsitteellisellä tasolla tarvitaan monipuolisempia työkaluja, jotka osaavat tulkita luettuja rakenteita.

Toinen ulottuvuus on ohjelman taso. Käyn tasot lyhyesti läpi, vaikka pääasiassa olenkin keskittynyt tietokantatason mallinnusmenetelmiin. Ohjelmassa muutosprosessi voidaan suorittaa välikerrosten, hakulauseiden tai logiikan tasolla. Yksinkertaisimmassa välikerrosstrategiassa luodaan ohjelma, joka toimii kuten entinen tietokanta, mutta muuntaa vanhanmalliset tietokantaoperaatiot uusiksi. Tällöin ohjelma siis toimii kuten ennenkin.

Hakulausetasolla muokataan ohjelman käyttämiä hakulauseita vastaamaan uutta tietokantaa. Logiikan tasolla taas muutetaan koko ohjelmaa, niin että se saa täyden hyödyn järjestelmämuutoksista.

Sijoittamalla kunkin menetelmän yhdelle sijalle ulottuvuusakseleilla saadaan siis yhteensä kuusi menetelmälajia.

7. Erilaiset tietokantojen takaisinmallinnusmenetelmät

Kirjallisuuskartoituksen lopuksi esittelen muutamia käytäntöjä mallintamisprosessin suorittamiseksi. Ensimmäisenä esittelen tavan jäsentää prosessia graafisesti ja kaksi jälkimmäistä ovat tapoja muuntaa relaatiorakenteet olioiksi.

7.1 Yleiset sumean päättelyn verkot takaisinmallinnuksen apuna

Tietokantojen takaisinmallintaminen perustuu eri lähteistä hankittaviin tietoihin. Koska nämä oletukset eivät ole varmoja, tarvitaan jokin keino niiden käsittelemiseen ja vertailuun. Jahnke et al. [1997] kehittivät tällaisen menetelmän pohjautuen yleisiin sumean päättelyn verkkoihin (generic fuzzy reasoning nets). Menetelmänsä kohdealueen he rajaavat oman jaottelunsa mukaisesti ajateltuna takaisinmallinnus-prosessin ensimmäiseen vaiheeseen eli tietokantakaavion takaisinmallinnukseen.

Erityisenä perusteena sumean päättelyn verkkojen puolesta kirjoittajat käyttävät takaisinmallintamisessa esiintulevia ristiriitaisia oletuksia, jotka seuraavat huonosta suunnittelusta ja itse ohjelman ristiriitaisuudesta. Käytettäessä verkkoesitysmuotoa ja oletuksien arvotusta saadaan mallintajalta kyselemällä tehokkaasti rakennettua järkevin mahdollinen kaaviokokonaisuus.

Kaikkia kannasta irtisaatavia oletuksia ei ole mahdollista ottaa mukaan heti, koska suuri vaihtoehtomäärä johtaisi liian suureen analysoitavaan verkkoon. Sen sijaan aloitetaan pienemmällä määrällä oletuksia ja lisätään niitä mallinnusprosessin edetessä.

Kirjoittajat kutsuvat oletuksia aksioomiksi (axiom) ja ryhmittelevät ne vahvoihin, heikkoihin ja muokkautuviin aksioomiin. Vahvat aksioomat ovat perusoletuksia, joita ei voida mallinnusprosessin edetessä todistaa vääriksi. Heikot aksioomat ovat mallintajan mukaan tuomia oletuksia. Ne voivat muuttua prosessin aikana ja niitä voidaan myös hylätä. Muokkautuvat aksioomat ovat oletuksia joita ei hyväksytä vahvoiksi, koska vahvojen aksioomien liian suuri määrä vaikeuttaisi mallinnusta. Myös nämä aksioomat muokkaantuvat mallinnusprosessin aikana.

Käsitellessä epätäydellistä ja muuttuvaa tietoa, tarvitaan vaikutusmekanismi, joka mahdollistaa ei-monotonisen (ihmisen päättelyä matkimaan pyrkivän) päättelyn. Kirjoittajat käyttävät tähän tarkoitukseen Petri-verkkoja niiden rinnakkaisuuden vuoksi. Heidän käyttämänsä algoritmi on seuraavanlainen:

Ensiksi sumea Petri-verkko v1 luodaan sumean päättelyn verkon v2 avulla. Jokaista verkon v2 propositiota kohden luodaan verkkoon v1 kaksi paikkaa, toinen proposition totuutta ja toinen epätotuutta varten. Ensin luodaan aluksi oletetut heikot ja vahvat aksioomat. Tämän jälkeen tutkitaan, onko uusien propositioiden myötä jonkin implikaation esiehdot täyttyneet. Myönteisessä tapauksessa luodaan implikaation seurauspropositiot. Tätä jatketaan, kunnes uusia propositioita ei enää synny.

Toisessa vaiheessa luodaan siirtymiä positiivisten ja negatiivisten paikkojen välille. Implikaation lisääminen johtaa ainakin kahden siirtymän luomiseen, sillä a => b johtaa siihen, että pätee myös ¬b => ¬a. Yleisesti ottaen siirtymien määrä on sama kuin

propositioiden edeltävässä laajennetussa implikaatiossa.

Kolmanneksi vahvoihin ja muokkautuviin aksioomiin johtavat siirtymät poistetaan, koska niiden oletuksia ei saa arvioinnin aikana muuttaa. Lopuksi on arviointivaihe, jossa aksioomien varmuus otetaan luotuihin propositioihin.

Esimerkiksi verkossa voi olla oletettuna {a => b, c => d, d => e}. Mukaan otetaan vahva aksiooma a, ja huomataan, että implikaation ”a => b” esiehto on täyttynyt, jolloin mukaan otetaan myös b. Aksiooman b mukaan ottaminen ei täytä uusia esiehtoja, joten siirrytään toiseen vaiheeseen.

Toisessa vaiheessa meillä on siis {a, b}. Näiden välille luodaan siirtymät a => b ja ¬b => ¬a. Kolmannessa vaiheessa huomataan, että siirtymä ¬b => ¬a johtaa vahvaan aksioomaan, minkä vuoksi se poistetaan ja jäljelle jää a => b. Arviointivaiheessa otetaan huomioon, että a on vahva aksiooma, joten siitä seuraava b on myös arvoltaan vahva.

7.2 Takaisinmallinnus olioiksi: Premerlani ja Blaha

Tietokantakaavion takaisinmallinnuksen eri vaiheita ovat tutkineet muun muassa Premerlani ja Blaha [1994]. He käyttivät pohjana SQL-mallia.

Kirjoittajien mukaan oliopohjaiset mallit ovat luonnollinen työkalu takaisinmallinnustyöhön. Tämä on tosiaan varsin selvä asia: oliot pystyvät kuvaamaan sen, mitä relaatiomallikin ja lisäksi oliomallit tarjoavat monipuolisempia merkintöjä.

Ensimmäinen askel on taulujen muuttaminen luokiksi ja sarakkeiden attribuuteiksi. Toiseksi etsitään avainehdokkaita ja kolmanneksi ovat vuorossa vierasavaimet. Moderneista tietokantajärjestelmistä vierasavaimet saa suoraan, vanhemmista järjestelmistä avaimet joutuu mallintaja etsimään. Vierasavainten etsintä tapahtuu ratkomalla ensin homonyymit ja synonyymit.

Samanlaiset attribuuttinimet, tietotyypit ja tietoalueet voivat olla merkki vierasavainsuhteesta. Yleistysten vuoksi tässä vaiheessa on vielä kyse vierasavainryhmistä, eikä yksittäisistä avaimista.

Vierasavainten etsimisen jälkeen Premerlani ja Blaha jatkavat jalostamalla alkuperäisiä luokkia. Horisontaalisesti pilkotut luokat, joilla on sama tietomalli pitää yhdistää. Näitä on jaettu useammiksi tauluksi ehkä tietokannan jakamiseksi useammalle palvelimelle tai muiden optimointien vuoksi. Kannattaa myös etsiä rajoitteita, jotka esitetään tauluina. Näistä vihjaa se, ettei tauluun ole yhdistetty vierasavainryhmiä.

Viidentenä vaihteena on yleistyksien huomaaminen. Kannattaa keskittyä isoihin vierasavainryhmiin ja myös toisen taulun pääavaimen kokonaan sisältävä vierasavain voi olla merkki yleistyksestä. Jos myös aliluokkien attribuutit on nostettu yliluokkaan, on seurauksena attribuuttiryhmiä, joissa joko kaikissa on jokin arvo, tai sitten kaikki ovat tyhjiä null-arvoja.

Tämän jälkeen voidaan keskittyä olioiden välisiin suhteisiin. Kahdesta vierasavaimesta koostuva luokka on selvä merkki suhteesta. Suhteen voi myös muodostaa olioiden välille, kun avainehdokas sisältää vierasavaimen ja ei-

vierasavainattribuutteja. Loput suhteet esitetään vierasavaimina. Premerlanin ja Blahan mukaan suhteiden minimi- ja maksimiastelukuja voi olla vaikeaa selvittää pelkän datan avulla, Soutouhan [1996] huomasi saman asian myöhemmin kirjoittamassaan artikkelissa. Lopuksi osa luoduista suhteista kannattaa yrittää muuttaa keräytymiksi (aggregation).

Viimeisenä vaiheena on optimointien poistamiseksi suoritettavat muutokset. Luokkia voi ensinnäkin muuttaa linkkiluokiksi (suhteeksi, jonka linkit voivat osallistua suhteisiin muiden luokkien välillä). Kevyet 1:1-suhteet voidaan mallintaa myös attribuuteiksi, ja N-asteen suhteita pitäisi yrittää muuttaa pienemmiksi mikäli mahdollista.

Ongelmana Premerlanin ja Blahan mukaan on, että takaisinmallinnusvaiheessa on hankalaa löytää minimi- ja maksimiasteita. Tässä voisikin olla hyvä tilaisuus käyttää Soutoun aikaisemmin käsiteltyä menetelmää hyväksi. Toisaalta myös transitiivisia suhteita on selkeyden vuoksi hyvä järjestellä. Esimerkiksi taulujen A ja B sekä B ja C välillä voi olla suhde, joka voi olla selkeämpää järjestää suhteeksi luokkien A ja B sekä A ja C välille.

Siistintävaiheessa on myös huomioitava, että tietokannat voivat kuvata samaa suhdetta osoittimilla suhteen molemmissa päissä. Onkin varottava, ettei luoda samaa suhdetta kaavioon useampaan kertaan. Luokkahierarkian selkeyttämiseksi on ehkä luotava uusia luokkia, joita tietokannassa ei mainita mitenkään.

Transitiivisuus pitää huomata myös yleistyksien kautta kulkevana. Ilman semanttista tulkintaa on vaikeaa tunnistaa kahden erillisen luokan välisen varsinaisen suhteen ja yleistys-erikoistus-suhteen eroa. Jos tietokannassa luokalla A vierasavainsuhde luokan B kaikkiin aliluokkiin, voidaan suhde siirtää koskemaan luokkaa B.

7.3 Takaisinmallinnus olioiksi: Ramanahan ja Hodges

Myös Ramanathan ja Hodges [1997] kehittivät menetelmän oliorakenteiden löytämiseksi relaatiotietokannasta. Menetelmä vaatii, että mallinnettava relaatiotietokanta on vähintään toisessa normaalimuodossa (se ei siis vaadi kolmatta normaalimuotoa, kuten useat muut). Lisäksi menetelmä vaatii esitietona pää- ja vierasavaimet. Myös homonyymi- ja synonyymiominaisuudet ovat tiukat: samaan arvoalueeseen viittaavat attribuutit oletetaan samoiksi ja homonyymejä ei oleteta olevan. Ramanahan ja Hodgeskin käyttivät SQL-mallia.

Ramanathan ja Hodges aloittavat määrittelemällä kiinnostaviksi rakenteiksi luokat ja suhteet. Ensinnäkin on löydettävä ne tietokantataulut, jotka viittaavat olioluokkaan. Toiseksi pitää havaita niiden väliset suhteet; näitä suhteita on kolmea tyyppiä: yhteys (association), periytyminen (inheritance) ja keräytymä (aggregation).

Menetelmä aloittaa tutkimalla tiedossa olevia riippuvuuksia ja muuntaen siten kaavion kolmanteen normaalimuotoon. Tällä tavalla muodostetut tietokantataulut mallinnetaan luokiksi lukuuottamatta niitä, joiden pääavain muodostuu kokonaan

vierasavaimista; jälkimmäiset mallinnetaan suhteiksi. Vierasavainten perusteella luodaan suhteita luokkien välille.

Periytymissuhteiden mallintamisessa Ramanathan ja Hodges ottavat huomioon samat tapaukset, kuin Premerlani ja Blaha: Yliluokka ja aliluokat voivat olla eri tauluja, joissa molemmissa käytetään samaa pääavainta. Toisaalta aliluokkien attribuutit on voitu kaikki tuoda yliluokkaan tai yliluokan attribuutit sisällyttää aliluokkiin.

Keräytymien havaitsemiseen Ramanathan ja Hodges ovat löytäneet kaksi tapausta. Ensimmäinen merkki keräytymästä on tapaus, jossa relaation R1 pääavaimessa on enemmän kuin yksi attribuutti, eivätkä kaikki noista attribuuteista liity vierasavaimeen. Relaatio R1 voi olla keräytymäsuhteessa relaatioon R2, jos R2 viittaa eniten relaation R1 pääavaimen vierasavaimiin.

Toisessa tapauksessa käytetään hyväksi relaatiotietokantojen kyvyttömyyttä säilöä lukutaulukoita. Tätä rajoitusta kierretään yleisesti käyttämällä binääritietotyyppiä, jonka tietokantaa käyttävä sovellus muuntaa hakiessaan taulukoksi ja tallentaessaan binäärimuotoiseksi. Koska ohjelmallisesti binääritiedon laadun tulkitseminen ei onnistu on binääritietoa kohdattaessa kysyttävä attribuutin merkitystä käyttäjältä.

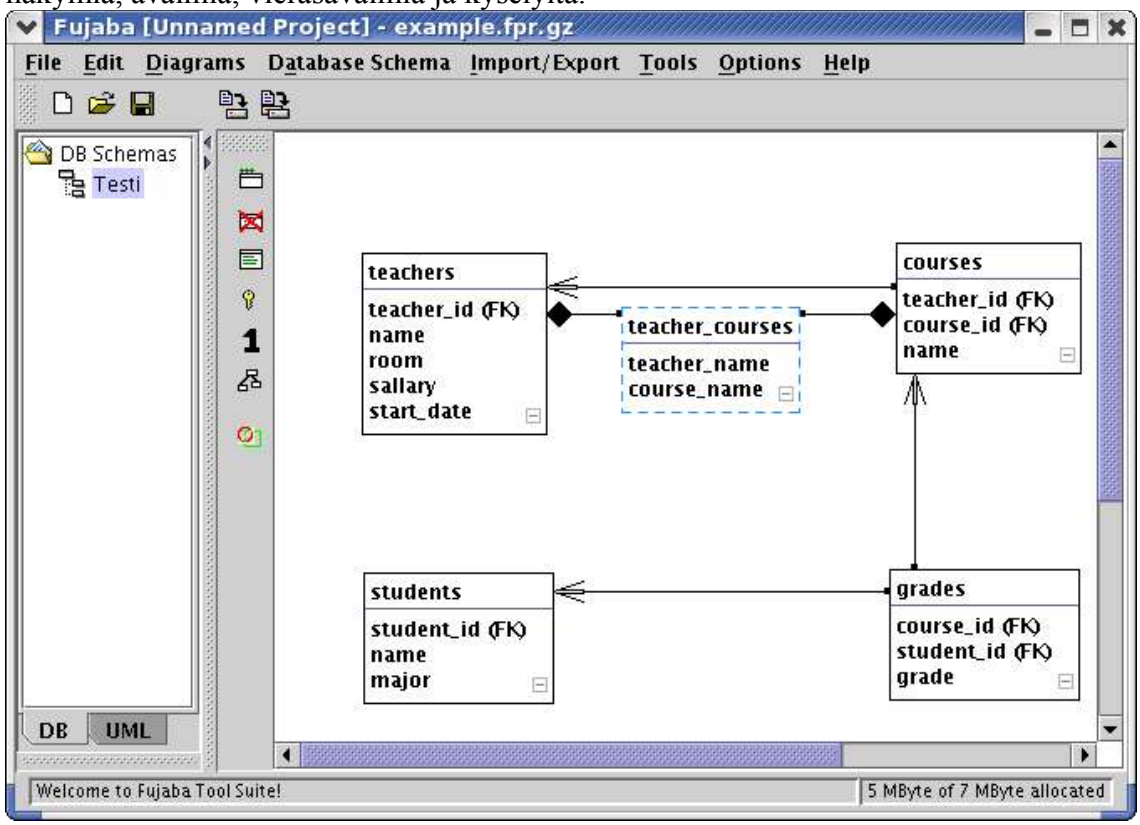
Ramanathanin ja Hodgesin tutkimus tuo siis uusina piirteinä Premerlanin ja Blahan menetelmään verrattuna ohjeita keräytymien havaitsemiseen sekä siihen liittyen huomion, että binäärimuotoinen tieto on hyödyllistä huomioida erityistapauksena. Mitään tekniikkaa binääritietoa varten Ramanathan ja Hodges eivät kuitenkaan kehittäneet (se olisikin ollut erittäin hankalaa), vaan merkityksiä pitää kysellä mallintajalta.

8. Sovellus: Fujaba

Takaisinmallinnuksen ensimmäisenä tietokantaan liittyvänä askeleena voidaan siis pitää tietokantakaavion lukemista johonkin helposti tulkittavaan ja muokattavaan muotoon. Graafisuus usein helpottaa kaavioiden käsittämistä, erityisesti vierasavainsuhteet on helpompi hahmottaa kuvana kuin tekstinä.

Tällaista tutkintaa voidaan suorittaa esimerkiksi Fujaba-ohjelmalla. Fujaba on alunperin UML-kaaviosuunnitteluun tarkoitettu Java-pohjainen työkalu, johon voi lisukemekanismin (plugin) avulla liittää ulkopuolista toimintaa. Tätä lisukerajapintaa käyttäen olen luonut Fujabaan tietokantatoiminnallisuutta [Seppi and Nummenmaa, 2003] (kuva 1).

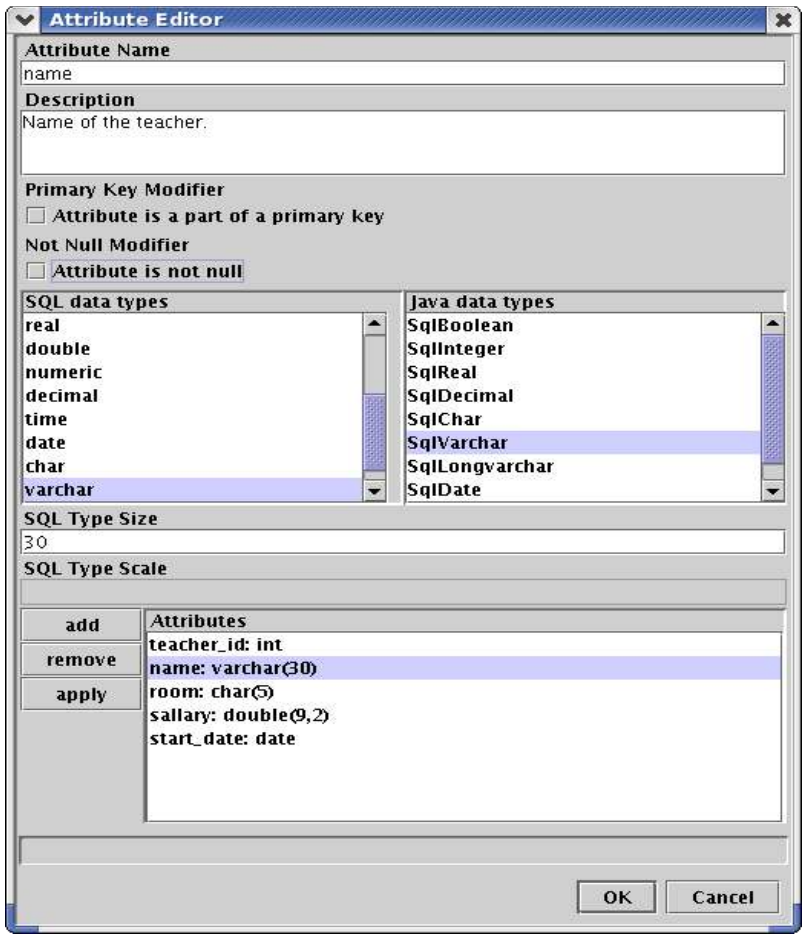
Tietokantalisuke tukee kaikkia yleisimipiä tietokantakäsitteitä, kuten tauluja, näkymiä, avaimia, vierasavaimia ja kyselyitä.



Kuva 1: Tietokantakaaviolisukkeen esimerkkinäkymä

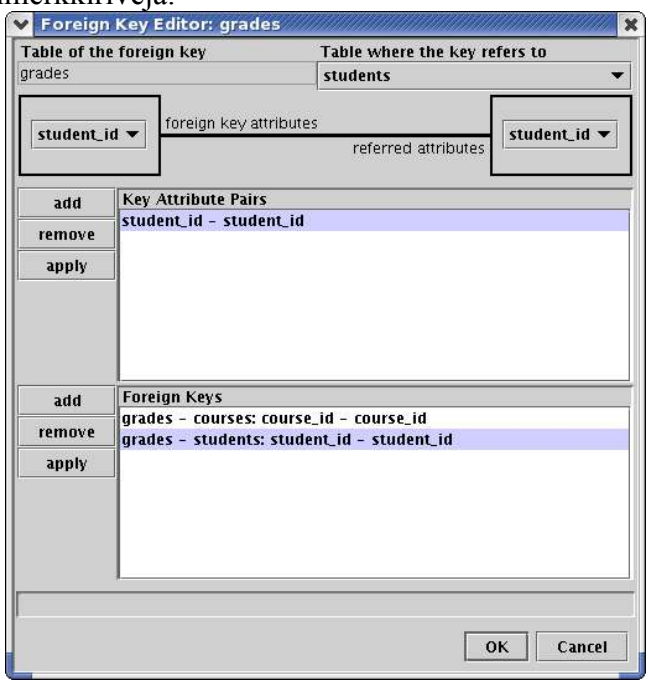
8.1 Taulut, näkymät ja kyselyt

Tauluille käyttäjä voi antaa nimen lisäksi kuvauksen ja pakettimäärityksen myöhemmin selitettävää Java-koodigenerointia varten. Tauluihin liittyy luonnollisesti myös attribuutteja, joiden ominaisuuksia ovat nimi, kuvaus, tieto siitä, voiko attribuutti saada null-arvoja tai onko se osa pääavainta sekä SQL-tietotyyppi, Java-tietotyyppi ja tietotyypistä riippuen mahdollisesti myös pituus ja desimaali. Myös Java-tietotyyppi on Java-koodigeneroinnin tarpeisiin (kuva 2).



Kuva 2: Attribuuttidialogi.

Attribuuteille voi myös määrittää esimerkkiarvoja. Tämä tapahtuu antamalla taulukohtaisesti esimerkkirivejä.



Kuva 3: Vierasavaindialogi

Vierasavaimia voidaan määritellä tauluihin liittyen omalla dialogillaan (kuva 3). Dialogissa määrättynä on vierasavaimen lähtötaulu, alasvetovalikosta voi valita tulotaulun ja molempien taulujen attribuutit, ja lisätä valitun attribuuttiparin ylempään

laatikkoon. Kun tarvitut attribuuttiparit on lisätty, voidaan valmis avain ottaa mukaan lisäämällä se alempaan laatikkoon. Avainten määrittely tapahtuu vierasavainmäärittelyn kaltaisesti, mutta nyt listataan vain yhden taulun attribuutit.

Eteenpäinmallinustoiminnallisuutena taulujen välisiä vierasavainsuhteita voi kysyä myös tietokantalisukkeelta, joka tutkii attribuuttien nimiä ja avainrakenteita ja yrittää niiden perusteella päätellä mahdollisia vierasavaimia. Arvatut vierasavaimet näytetään dialogissa ja käyttäjä voi niistä valita haluamansa luotaviksi.

Toinen taulukohtainen eteenpäinmallinnustyökalu on taulujen liittäminen ja dekompositio. Liittäminen kokoaa kahden vierasavaimella yhdistetyn taulun attribuutit yhdeksi ja päivittää vierasavainsuhteet. Tauluista lähtevät vierasavaimet päivitetään normaalisti vain lähtemään uudesta taulusta, samoin kuin lähtötauluun päätyvät vierasvaimet. Tulotauluun päätyvät vierasvaimet päivitetään, vain jos vierasavainsuhteen attribuutit myös lähtötaulussa muodostavat avaimen. Muussa tapauksessahan ei voida luottaa siihen, että vierasavain todellakin edelleen yksilöisi taulun tietyn rivin.

Dekompositio-operaatiossa vastaavasti erotetaan yksi taulu kahdeksi. Lähtevät ja tulevat vierasavaimet luodaan siihen tauluun, jossa on avaimen kaikki attribuutit, jos kaikki attribuutit on molemmissa, luodaan siis kaksi uutta vierasavainta. Tämä vähentää tietokantamallintajan työtä normalisointivaiheessa.

Näkymiä voi määritellä osittain samalla tavalla kuin taulujakin. Yleisinä ominaisuuksina kullekin näkymälle määritetään nimi, paketti ja kuvaus. Attrbuutteja voi lisäillä valiten niitä kaavion taulujen attribuuteista, mukaanotetuille attribuuteille voi antaa uuden, näkymäkohtaisen nimen. Lisäksi käyttäjän on annettava näkymänluontilauseen WHERE-lauseke.

Tietokantalisukkeessa on myös etsintätoiminnallisuus taulujen ja näkymien sekä attribuuttien etsimistä varten. Attribuuttien etsintätoiminnallisuus tukee jokerimerkkejä.

Kyselyissä on monia samanlaisia piirteitä näkymien kanssa. Kyselyiden ominaisuuksiin kuuluvat nimi, paketti, WHERE-lauseke ja taululiitokset. Käyttäjä voi valita haluamansa taululiitokset ja jos taulujen välillä on vierasavainsuhde, lisää tietokantalisuke automaattisesti vierasavaimen mukaisen liitoksen WHERE-lausekkeeseen. Käyttäjä voi silti itse asettaa erityisiä ehtoja ja muutenkin muokata WHERE-lausetta vapaasti.

Tietokantakaavioille ei ole tarkasti vakiintunutta piirtokäytäntöä, joten tietokantalisuke käyttää Fujaban UML-piirtokäsitteistöä. Taulu on yhtenäisellä viivalla piirretty suorakulmio, jonka vaakaviiva jakaa nimi- ja attribuuttiosioihin. Näkymä on muuten samanlainen, mutta ulkoreuna on katkoviivaa. Näkymästä lähtee viiva kaikkiin tauluihin, joiden attribuutteihin se viittaa. Viivan taulunpuoleisessa päässä on salmiakkikuvio.

Kysely taas kuvataan kulmiltaan pyöristettynä nelikulmiona, jonka sisällä on nimi. Kyselykin on viivoilla yhdistetty tauluhin, joihin se viittaa. Vierasavaimet taas ovat nuolia, jotka osoittavat lähtötaulusta kohdetauluun.

8.2 Tietokannan tuonti kaavioksi

Takaisinmallinusta tukemassa lisäosa sisältää SQL-lauseiden jäsennystoiminnallisuuden, joka luo tietokantakaavion valituista SQL-lausetiedostoista. Tätä mahdollisuutta voidaan käyttää, jos kaavion ajantasallaolevat luontilauseet löytyvät, tai jos ne saadaan kannasta.

Toinen mahdollisuus kaavion uuttamiseen on kannan JDBC-ajurin kannasta antaman metatiedon käyttäminen. Tämä tapa puolestaan edellyttää, että kantaan on olemassa JDBC-ajuri, ja että se tukee tarpeeksi suurta osajoukkoa DatabaseMetaData -luokan metodeista. Koska JDBC-ajuri usein on puutteellinen, ei metatiedon käyttäminenkään toimi virheettömästi. Esimerkiksi näkymiä ei välttämättä voida ottaa mukaan, koska JDBC-ajuri ei kerro, mihin tauluattribuutteihin näkymäattribuutit viittaavat.

8.3 Kaavion vienti tietokannaksi

Kun kanta on valmiiksi suunniteltu, pitää kaavio muuttaa tietokannaksi. Tämä voidaan tehdä tietokantalisukkeella kahdella tapaa. Tietokantakaavion voi muuttaa SQL-lausetiedostoiksi tai sen voi siirtää suoraan tietokantajärjestelmään.

Tietokannan SQL-lauseiksi muutamiseen tietokantalisuke käyttää Nummenmaan alunperin tekstiedostosyötteitä lukenutta DBSWTOOL-työkalua, johon tein eräitä tietokantalisukkeen vaatimia lisäyksiä [Seppi and Nummenmaa, 2003]. Työkalu luo kolmea tulostetyyppiä: HTML-, SQL- ja Java-hakemistot.

HTML-tiedostoissa kuvataan taulut, kyselyt, näkymät ja attribuutit. SQL-hakemistossa taas on jokaisen taulun luontilause erikseen omissa tiedostoissaan ja all.sql-tiedostossa järjestettynä kaikki luontilauseet niin, ettei niiden syöttäminen tietokantaan sellaisenaan aiheuta vierasavainviittausristiriitoja.

SQL-kieli vaihtelee tietokantakohtaisesti, esimerkiksi koska valmistajat ovat halunneet tarjota ominaisuuksia, joita ei SQL-standardiin ole vielä kuulunut. Tämän vuoksi samat SQL-luontilauseet eivät toimi kaikissa kaikkien kantojen SQL-murteissa tai eri kantoja varten erikseen suunnitellut lauseperheet toimivat standardia SQL-kieltä paremmin.

DROP TABLE -käskyllä ei standardin mukaan ole IF EXISTS -määrettä, mutta osa tietokantavalmistajista on halunnut tarjota sellaisen. Tuo määre on hyödyllinen automaattisesti tuotettaviin SQL-lauseisiin, sillä ei voida tietää, käytetäänkö niitä luomaan tauluja tyhjään kantaan vai muuttamaan jo luotujen taulujen määrittelyitä.

Tämän murreongelman vuoksi tietokantalisuke kysyy käyttäjältä tietokannan tyypin. Tuettuja tyyppejä tällä hetkellä ovat MySQL, HSQLDB ja standardi SQL-kieli (PostgreSQL).

Java-hakemistossa taas on pakettiperustainen alirakenne, jokaiselle paketille on oma hakemistonsa, jonne kuuluvat Java-tiedostot on sijoitettu [Seppi and Nummenmaa, 2003]. Nämä Java-tiedostot ovat SQL-lauseiden perusteella luodun tietokannan käsittelyyn tarkoitetun ohjelman luuranko.

Jokainen taulu ja näkymä tuottaa kaksi Java-tiedostoa: Db[luokan nimi].java ja [luokan nimi].java [Seppi and Nummenmaa, 2003]. Ensiksimainittu sisältää automaattisesti luotavan tietokantaa käsittelevän osuuden. Tähän osuuteen kuuluu attribuuttien arvojen asettamisen lisäksi tietojen kysely kannasta. Jälkimmäinen luokka taas perii ensiksimainitun, mutta ei sisällä automaattisesti luotavaa toiminnallisuutta, vaan on sovelluskohtaisesti kirjoitettavaa ohjelmakoodia varten tarkoitettu.

Myös kyselyistä tehdään Db[kyselyn nimi].java ja [kyselyn nimi].java -kaavaa noudattavat luokat. Kyselyt eroavat kuitenkin näkymistä siinä, että ne palauttavat tuloksena relaatioilmentymiä, joita käyttäjä voi sitten päivittää tarpeen mukaan [Seppi and Nummenmaa, 2003].

Tietotyyppeinä käytetään Javan yleistyyppien sijaan itsetehtyjä tyyppiluokkia, Javan valmiita luokkia ei voida käyttää niiden epäyhtenäisyyden vuoksi [Seppi and Nummenmaa, 2003]. Primitiivisiä tietotyyppejä ei voida käyttää, koska ne eivät voi saada null-arvoa, ja oliotyypeille puolestaan ei ole olemassa yhteneviä rakentimia tai asettimia eikä myöskään yhtenevää virhehallintaa.

Toinen tapa on siis kaavion siirtäminen suoraan tietokantaan JDBC-ajurin avulla. Tällä hetkellä tuota muunnosta varten on hyvin yksinkertainen algoritmi:

Kuva 4: Kaavio-tietokantamuutosdialogi.

Ensin vertaillaan taulujen nimiä ja tutkitaan, löytyvätkö samat taulut uudesta ja vanhasta kaaviosta (vanha kaavio on muunnettava tietokanta, uusi puolestaan Fujaballa luotu kaavio, jollaiseksi tietokanta halutaan muuttaa). Seuraavaksi merkitään lisättäviksi taulut, jotka löytyvät vanhasta, mutteivät uudesta kaaviosta, ja poistettavaksi taulut, jotka löytyvät vanhasta, mutteivät uudesta kaaviosta. Muuttumattomat taulut listataan erikseen.

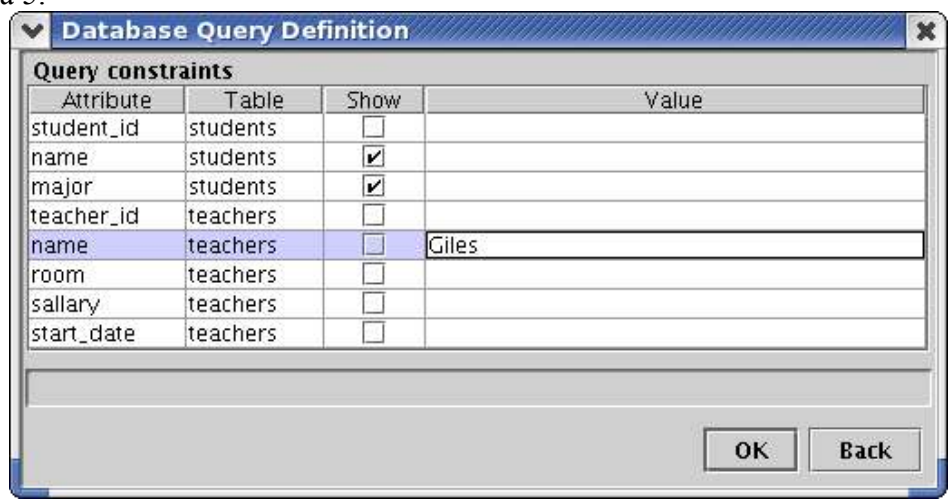
Taulujen läpikäynnin jälkeen käyttäjälle näytetään dialogi, jossa uudet taulut ja poistettavat taulut ovat yhdellä rivillä omissa listoissaan ja muuttumattomat taulut omalla rivillään omassa listassaan. Muunnosoperaatiota suorittava käyttäjä voi valita vastaavat taulut lisättävien ja poistettavien listasta ja merkitä ne samoiksi, mikä aikaansaa uudelleennimeämisoperaation. Uudelleennimeämisoperaatioita voi myös purkaa.

Taulujen attribuutit käyvät läpi saman tutkinnan kuin taulut. Tutkitaan siis tauluja, joille löytyy samanniminen vastinpari ja tauluja, joille käyttäjä on määrännyt uudelleennimeämisoperaation. Oletuksena uudesta ja vanhasta taulusta löytyvät attribuutit pidetään, vain uudesta löytyvät lisätään ja vain vanhasta löytyvät poistetaan. Erona on vain, että attribuuttien kohdalla nimen lisäksi tutkitaan myös tietotyyppejä. Käyttäjä voi attribuutteihin tehdä uudelleennimeämis- ja tyypinvaihtomäärityksiä.

Menetelmä siis on hyvin yksinkertainen ja suunnitteilla onkin lisätä siihen lisätoiminnallisuutta ja monimutkaisempi algoritmi. Eräs lisätoiminnallisuus voisi olla tietokantaevoluution huomioiva normalisointi- ja denormalisointioperaatioita etsivä algoritmi, jota esittelen tarkemmin myöhemmin.

8.4 Kokeilutietokanta

Tietokannan kokeilemista varten lisukkeen mukana on HSQLDB-tietokanta. HSQLDB soveltuu tarkoitukseen erityisen hyvin, koska se on Java-tietokanta, ja toimii siis todennäköisesti kaikkialla missä Fujabakin. Näin ollen ei tietokannan vuoksi tarvitse tehdä eri paketteja eri käyttöjärjestelmille. HSQLDB on lisäksi kevyt, mutta silti kohtuullisen kehittynyt. Kokeilutietokannan kyselynmuodostusdialogi on esitetty kuvassa 5.



Kuva 5: Kokeilutietokannan kyselynmuodostusdialogi.

Lisuke osaa käyttää HSQLDB-tietokantaa automaattisesti ilman, että käyttäjän tarvitsee antaa mitään asetustietoja tai asentaa tietokantajärjestelmää erikseen. Käyttäjä voi valita valikosta tietokannan luomisen, minkä jälkeen tietokantalisuke tekee kaaviosta HSQLDB-tietokannan. Kantaan syötetään taulut, näkymät ja käyttäjän antamat esimerkkitiedot.

Käyttäjä voi tehdä hakuja tietokantaan, eli tarkemmin käyttäjän taulukohtaisesti

kaavioon antamiin esimerkkitietoihin. Haut tapahtuvat joko avustajatyylisen dialogisarjan avulla tai kirjoittamalla SQL-kyselyn. Dialogisarjassa käyttäjältä kysytään ensin, minkä taulujen tietoja hän haluaa esille tai minkä taulujen attribuutteihin liittyen hän haluaa antaa hakuehtoja. Tauluja voi valita useampia, mutta näkymiä vain yhden kerrallaan, koska näkymän ja taulun sitominen samaan SQL-kyselyyn on automatisoinnin kannalta hankalaa.

Tauluvalinnan jälkeen käyttäjälle annetaan taulukkodialogi, jossa on lueteltuna valittujen taulujen attribuutit sekä niiden taulut. Käyttäjä voi muokata tiedon siitä, näkyykö attribuutin arvot hakutuloksissa ja hakuehto (kuva 5). Lisuke osaa itse tehdä vierasavainten pohjalta liitosehdot WHERE-lausekkeeseen, mikäli liitosehto on vierasavainten pohjalta luotavissa. Tämä on mahdollista ovatpa taulut yhteydessä suoralla vai muiden taulujen kautta kulkevalla vierasavainliitoksella. Hakutulokset näytetään taulukkona, jonka voi tallentaa HTML-sivuksi.

SQL-kyselyn puolestaan voi syöttää tekstikenttään, minkä jälkeen lisuke suorittaa haun ja rakentaa tuloksista samanlaisen taulukon kuin dialogihaussakin.

8.5 Normalisointi- ja denormalisointioperaatiot

Eräs tietokantojen takaisinmallinnuksen lähestymistapa on ottaa huomioon tietokantaevoluutio. Tietokantamuutosten merkittäviin lajityyppeihin kuuluvat normalisointi ja denormalisointi, joita enimmäkseen toki tehdään tietokantaa suunnitellessa ennen käyttöönottoa. Saatetaan niitä silti tehdä myös käyttöönoton jälkeen.

Normalisointi siis tarkoittaa yhden taulun jakamista kahdeksi. Syynä operaatioon on yleensä jonkin tai joidenkin attribuuttien riippuvuus taulun jostain attribuuteista, jolloin riippuvat attribuutit poistetaan vanhasta taulusta ja lisätään uuteen, myös riippuvuutta aiheuttavat attribuutit kopioidaan uuteen tauluun pääavaimeksi. Operaatio siis nimensä mukaisesti kasvattaa tietokantakaavion normaalimuotoastetta.

Toisaalta jos jokin normalisointi havaittiin turhaksi tai muusta syystä halutaan kertätä kahden taulun attribuutit yhdeksi tauluksi, saatetaan suorittaa denormalisointi. Syynä denormalisointiin lienee usein halu yksinkertaistaa SQL-kyselyitä. Operaatio päinvastoin kuin normalisointioperaatio. Näitä operaatioita hyväksikäyttää tietokantamuunnosalgoritmi, jonka tarkoitus on muutaa vanha tietokanta vastaamaan uutta tietokantakaaviota, jotta tietokannan tietoja hukkuisi mahdollisimman vähän. Jos jokin taulu on uudesta tietokantakaaviosta vain poistettu, muutos tietenkin aiheuttaa tietohävikkiä, mutta algoritmin tavoitteena onkin havaita tilanteet, joissa attribuutteja on vain siirretty muualle normalisoinnin ja denormalisoinnin keinoin.

Algoritmi perustuu pääavaineroavaisuuksiin [Nummenmaa and Seppi, 2004]. Alkutilanteessa on kaksi kaaviota, uusi ja vanha. Tarkoitus on siis muuttaa vanha kaavio vastaamaan uutta. Algoritmi aloittaa vertailemalla taulujen pääavaimia. Jos uudesta ja vanhasta taulusta löytyvien pääavainten attribuuttien nimet ja tyypit ovat samat, poistetaan molemmat pääavaimet pääavaineroavaisuuksista, ja merkitään löydetyt taulut pariksi.

Tämän jälkeen alkaa ensimmäinen sovittava osa: pääavainsovitus [Nummenmaa and Seppi, 2004]. Jos on mahdollista saada kaksi sovittamatonta pääavainta samoiksi lisäämällä tai poistamalla toisesta attribuutteja, muodostetaan näistä pääavaimista (tauluista) pari ja poistetaan kyseiset pääavaimet pääavaineroavaisuuksista. Kun kaikki pääavaimet on näin käyty läpi ja jäljellä on vielä pääavaineroavaisuuksia, alkaa algoritmin normalisointia ja denormalisointia etsivä osa.

Algoritmi käy läpi kaikki pääavainten perusteella löytyneet parit [Nummenmaa and Seppi, 2004]. Jos vanhassa taulussa on attribuutteja, joita uudesta ei löydy, epäillään, että tietokannassa on suoritettu normalisointioperaatio. Epäilyn vahvistamiseksi tai kumoamiseksi käydään läpi kaikki uudesta taulusta lähtevät vierasavaimet. Jos haettuja attribuutteja löytyy vierasavainviitatuista tauluista, päättelee algoritmi, että on tapahtunut normalisointi, minkä seurauksena vanhan kaavion taulu on muuttunut kahdeksi tauluksi uudessa kaaviossa. Tällöin löydetyn attribuutin taulun pääavain poistetaan pääavaineroavaisuuksien joukosta. Jos kaikkia uudesta taulusta puuttuvia attribuutteja ei ole vielä löydetty, jatkaa algoritmi käymällä läpi loput uudesta taulusta lähtevät vierasavaimet.

Seuraavassa vaiheessa käydään pääavainparit läpi toisen kerran [Nummenmaa and Seppi, 2004]. Tällä kertaa tutkitaan, onko uudessa taulussa attribuutteja, joita ei vanhansta taulusta löydy. Jos näin on, epäillään denormalisointioperaatiota. Epäily vahvistetaan tai kumotaan käymällä läpi vanhan taulun vierasavainsuhteet samoin kuin normalisointioperaatioita tutkittaessa käytiin läpi uuden taulun vierasavaimet. Vierasavainpoikkeavuuslistalta poistetaan kaikki taulut, joista uuden taulun attribuutteja löytyy.

Jos vielä tämän vaiheen jälkeen on pääavaineroavaisuuksia, algoritmi päättelee, että ne ovat uusia ja poistettaviksi tarkoitettuja tauluja [Nummenmaa and Seppi, 2004]. Näin siis uuden kaavion eroavat taulut luodaan tietokantaan ja vanhan kaavion taulut poistetaan. Seuraavaksi algoritmi suorittaa kantaan havaitsemansa normalisointi- ja denormalisointioperaatiot. Kuudentena vaiheena on poistaa tietokannasta vanhan kaavion attribuutit, joita ei uudesta kaaviosta löydy ja lisätä ne uuden kaavion attribuutit, joita vanhasta kaaviosta ei löydy. Lopuksi muutetaan (ei-pää)avain- ja vierasavainrajoitteet vastaamaan uutta kaaviota. Algoritmin toimintaa on havainnollistettu esimerkissä 1.

Esimerkki 1

```
Tietokannan taulut:
t1(a1#, a2, a3, a4)
t2(a1#, a2#, a6, a7)
t3(a2#, a5)
t4(a2#, a8#, a9, a10)
t5(a8#, a11#, a12)
t6(a11#, a13)
t7(a12#, a14#, a15)

Tietokantakaavion taulut
k1(a1#, a2#, a3#, a4)
```

```
k2(a1#, a2#, a5, a6, a7)
k3(a2#, a8#, a10)
k4(a8#, a9)
k5(a8#, a11#, a12)
k6(a12#, a16)
k7(a12#, a14#, a15#)

Alkutilanne:
Tietokannan vanhat taulut
{t1, t2, t3, t4, t5, t6, t7}
Kaavion uudet taulut
{k1, k2, k3, k4, k5, k6, k7}
Yhdistetyt taulut
{}

Vaihe 1: Samojen avainten tunnistaminen
Tietokannan poistettavat taulut
{t1, t3, t6, t7}
Kaavion yhdistetyt taulut
{k1, k4, k6, k7}
Yhteiset taulut
{t2 => k2, t4 => k3, t5 => k5}

Vaihe 2: Taulujen täsmäys pääavaimia muuttamalla
Tietokannan poistettavat taulut
{t3, t6}
Kaavion yhdistetyt taulut
{k4, k6}
Yhteiset taulut
{t2 => k2, t4 => k3, t5 => k5, t1 => k1, t7 => k7}

Vaihe 3: Normalisointien etsintä
Tietokannan poistettavat taulut
{t3, t6}
Kaavion yhdistetyt taulut
{k6}
Yhteiset taulut
{t2 => k2, t4 => k3&k4, t5 => k5, t1 => k1, t7 => k7}

Vaihe 4: Denormalisointien etsintä
Tietokannan poistettavat taulut
{t6}
Kaavion yhdistetyt taulut
{k6}
Yhteiset taulut
{t2&t3 => k2, t4 => k3&k4, t5 => k5, t1 => k1, t7 => k7}

Operaatiot:
Poistetaan taulu t6.
Luodaan taulu k6.
Päivitetään taulujen t1 ja t7 pääavaimet.
Muutetaan taulu k4 tauluiksi k3 ja k4.
Yhdistetään taulut t2 ja t3 tauluksi k2.
```

8.6 Näkymät ja oliorajapinnat

Jotta kantayhteensopivuus säilyisi mahdollisimman pitkään, eteenpäinmallinnusprosessi voi luoda vanhaa tietokantaa vastaavat näkymät tai muun rajapinnan muuttuneeseen tietokantaan. Henrardin ja muiden [2002] menetelmäjaossa nämä sijoittuvat ohjelmaulottuvuuden kaikkein yksikertaisimpaan lokeroon. Tällöin vanhat ohjelmat voisivat lukea tietoja kannasta kuten ennenkin. Tietojen päivittämisen, poistamisen tai lisäämisen mahdollisuus luonnollisesti riippuu käytetystä tavasta.

Näkymien luominen on tavoista yksinkertaisempi. Näkymien lisäämisen lisäksi ei tarvita muutoksia tietokantaan, eikä myöskään ohjelmaan tarvitse koskea. Kaikki toimii

kuten ennenkin.

Näkymien luominen edellyttää kuitenkin, että merkittävästi muuttuneet taulut uudelleennimetään. Taulun uudelleennimeäminen on siis välttämätöntä, jos taulun attribuutteja poistetaan tai uudelleennimetään, koska sen jälkeen taulua ei voi enää käyttää kuten ennen. Tietotyypin muutokset puolestaan eivät välttämättä riko yhteensopivuutta, eivätkä siis vaadi näkymän luomista.

Esimerkiksi tekstitietotyypin suurimman sallitun pituuden muutos ei vaikuta tietokannan hakutuloksiin, eikä päivitykseenkään, ellei muutoksia tehdä siten, että päivitys vanhasta ohjelmasta käsin ei enää onnistu. Jos tietotyyppimuutokset taas rikkovat yhteensopivuuden, ei sitä pelkkien näkymien avulla saa korjattua, ellei tietokanta tue tietotyyppimuunnoksia näkymämäärittelyssä. Ja vaikka tuki tietotyyppimuunnoksille löytyisikin, on toki vaarana attribuutin merkityksen muuttuminen tietotyypin mukana. Tällöinkään näkymistä ei ole apua.

Attribuuttien lisääminen tauluun ei vaikuta vanhojen ohjelmien toimintaan tiedonhaussa, mutta päivitys-, poisto- ja lisäysoperaatioita vanhaa tietomallia käyttävät ohjelmat eivät luotettavasti voi suorittaa. Näkymän hyödyt saadaan siis pelkästään poistamalla ohjelmalta päivitysoikeus tauluun.

Jos prosessi halutaan automatisoida, on mahdollista kehittää algoritmi nimeämään muutetut taulut automaattisesti uudelleen, jolloin näkymä voi käyttää taulun vanhaa nimeä. Tällainen automaattisesti toteutettu toiminnallisuus olisi siksikin hyvä, että se pystyisi tallentamaan muutoshistorian, mikä voitaisiin liittää tietokannan dokumentaatioon. Tuollainen käytäntö kuitenkin luultavasti häiritsisi käyttäjää liikaa, koska ohjelma ei pysty antamaan tauluille parhaita mahdollisia nimiä. Käytäntöä voisikin kehittää huomauttamalla käyttäjälle tarvittavista uudelleennimeämisistä ja antaa käyttäjän valita nimet.

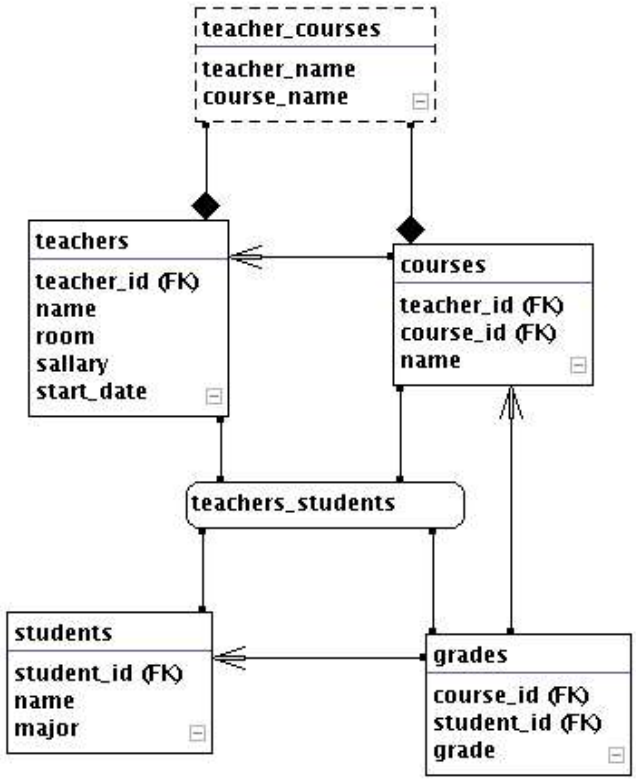
Näin automatisoitu näkymägenerointi pystyy ratkaisemaan osan taaksepäinyhteensopivuuden ongelmista, ja jos vanhaa tietokantamallia käyttävä ohjelma vain lukee eikä päivitä tietokannan tietoja, toiminee näkymäratkaisu yhtä hyvin kuin tietokantakin.

Kuvattujen ongelmien vuoksi näkymät eivät kuitenkaan ole millään lailla yleispätevä ratkaisu. Toinen varsinaiseen tietokantasisältöön koskematon ratkaisu on muuttaa tietokantaa käsittelevää ohjelmaa vastaamaan uutta tietokantaa. Tämä muuttaminen on kuitenkin hankalaa ja virhealtista. Sitä voidaan kuitenkin helpottaa, jos jo ohjelmaa tehtäessä on kiinnitetty mahdollisiin tietokantamuutoksiin huomiota.

Ohjelman muuttaminen uutta tietokantaa vastaavaksi käy parhaiten oliorajapinnalla, eli ohjelman tietomallin ja tietokannan väliin tehdyllä kerroksella, joka siis mahdollistaa ettei varsinaista ohjelmaa joudutakaan muuttamaan. Tällainen rajapinta on näkymiä hankalampi luoda, mutta hyvänä puolena sillä pystyy tekemään eri muutoksia rajattomasti: koska rajapinta on ohjelmakoodia se luonnollisesti pystyy käsittelemän tiedon oikeaan muotoon, jos tarvittavat tiedot vielä löytyvät tietokannasta.

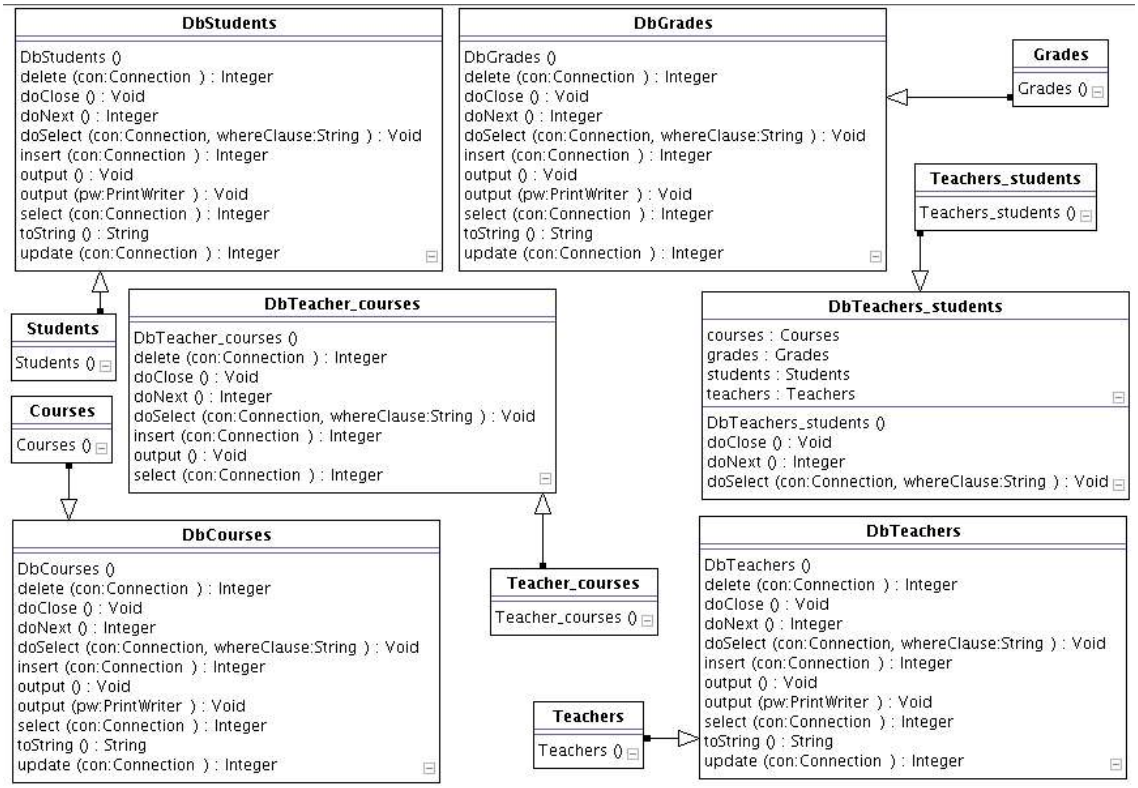
8.7 Eteenpäinmallinnusesimerkki

Esimerkki 2 kuvaa Fujaban tietokantakaaviolisukkeen kaavion tuontia tiedostoiksi. Esimerkin kaaviossa on neljä tietokantataulua: *teachers(teacher_id#, name, room, sallary, start_date), courses (teacher_id, course_id#, name), grades(course_id#, student_id#, grade)* ja *students(student_id#, name, major)*. Näiden välillä on vierasavainsuhteet, courses-teachers, grades-courses ja grades-students. Kaaviossa on myös näkymä *teacher_courses(teacher_name[teachers(name)], course_name[courses (name)])*, johon on koottu *teacher_name* -attribuutti teachers-taulusta ja *course_name* -attribuutti courses-taulusta, ja *teachers_students* -kysely. Esimerkki 2 liittyy kuvan 6 esimerkkitietokantakaavioon.



Kuva 6: Esimerkkitietokantakaavio.

Generoitujen Java-koodien luokkakaaviosta näkyy kaikille Db-luokille luotavat delete (Connection)-, doClose()-, doNext()-, doSelect(Connection, String)-, insert (Connection)-, output()-, output(PrintWriter), select(Connection)-, toString()- ja update (Connection) -metodit. Luokkakaaviossa (kuva 7), ei näy attribuuttien asetus- ja lukumetodeita.



Kuva 7: Tietokantakaaviosta muodostettujen Java-koodien luokkakaavio.

Esimerkki 2 varsinaisesta Db-tyypin Java-tiedostosta, DbCourses-java. Tiedostossa on ensin listattuna attribuutit sekä ohjelmallisina apumuuttujina PreparedStatement- ja ResultSet-oliot. Rakentimessa alustetaan muuttujat ilman arvoa (varsinaisten arvojen asettaminen tapahtuu myöhemmin) ja output- ja toString-metodit tuottavat erilaisia tekstiesityksiä.

Esimerkki 2

```
package fi.uta.cs.students;

import java.io.PrintWriter;

import java.sql.*;

import fi.uta.cs.sqldatatypes.*;

public class DbCourses {
    private SqlInteger teacher_id;
    private SqlInteger teacher_id_temp;
    private SqlInteger course_id;
    private SqlInteger course_id_temp;
    private SqlVarchar name;
    private PreparedStatement ps;
    private ResultSet rs;

    public DbCourses() {
         teacher_id = new SqlInteger();
         teacher_id_temp = new SqlInteger();
         course_id = new SqlInteger();
         course_id_temp = new SqlInteger();
         name = new SqlVarchar(30, null);
         teacher_id.setPrime(true);
         course_id.setPrime(true);
    }

    public String toString() {
         StringBuffer sb = new StringBuffer();
```

```
        sb.append("courses\n");
        sb.append("teacher_id:" +  teacher_id.toString() + "\n");
        sb.append("course_id:" +  course_id.toString() + "\n");
        sb.append("name:" +  name.toString() + "\n");
        sb.append("\n");
        return(sb.toString());
    }

    public void output() {
        System.out.println("courses");
        System.out.println("teacher_id:" +  teacher_id.toString());
        System.out.println("course_id:" +  course_id.toString());
        System.out.println("name:" +  name.toString());
        System.out.println();
    }

    public void output(PrintWriter pw) {
        try { pw.write(toString()); }
        catch (Exception e) {System.out.println(e);}
    }

    public int setTeacher_id(String input) {
        return teacher_id.setAndCheck(input, false, "teacher_id");
    }

    public SqlInteger getTeacher_id() {
        return teacher_id;
    }


    public int setCourse_id(String input) {
        return course_id.setAndCheck(input, false, "course_id");
    }

    public SqlInteger getCourse_id() {
        return course_id;
    }


    public int setName(String input) {
        return name.setAndCheck(input, false, "name");
    }

    public SqlVarchar getName() {
        return name;
    }

    public int insert(Connection con) throws SQLException {
        try {
            String prepareString = "insert into  courses values
(?,?,?)";
            PreparedStatement ps = con.prepareStatement
(prepareString);
            ps.setObject(1, teacher_id.getJdbc());
            teacher_id_temp.setAndCheck(teacher_id.toString(), false,
"teacher_id_temp");
            ps.setObject(2, course_id.getJdbc());
            course_id_temp.setAndCheck(course_id.toString(), false,
"course_id_temp");
            ps.setObject(3, name.getJdbc());
            int rows = ps.executeUpdate();
            ps.close();
            return rows;
        }
        catch (SQLException e) {
            throw e;
        }
    }

    public int update(Connection con) throws SQLException {
```

```
        try {
            String prepareString = "update  courses" +
        " set teacher_id = ?, course_id = ?, name = ?" +
            " where teacher_id = ? AND course_id = ?;";
            PreparedStatement ps = con.prepareStatement
(prepareString);
            ps.setObject(1, teacher_id.getJdbc());
            ps.setObject(2, course_id.getJdbc());
            ps.setObject(3, name.getJdbc());
            ps.setObject(4, teacher_id_temp.get());
            ps.setObject(5, course_id_temp.get());
            int rows = ps.executeUpdate();
            ps.close();
            teacher_id_temp.setAndCheck(teacher_id.toString(), false,
"teacher_id_temp");
            course_id_temp.setAndCheck(course_id.toString(), false,
"course_id_temp");
            return rows;
        }
        catch (SQLException e) {
            throw e;
        }
    }

    public int delete(Connection con) throws SQLException {
        try {
            String prepareString = "delete from courses" +
            " where teacher_id = ? AND course_id = ?;";
            PreparedStatement ps = con.prepareStatement
(prepareString);
            ps.setObject(1, teacher_id.getJdbc());
            ps.setObject(2, course_id.getJdbc());
            int rows = ps.executeUpdate();
            ps.close();
            return rows;
        }
        catch (SQLException e) {
            throw e;
        }
    }

    public int select(Connection con) throws SQLException {
        try {
            String prepareString = "select * from  courses" +
            " where teacher_id = ? AND course_id = ?;";
            PreparedStatement ps = con.prepareStatement
(prepareString);
            ps.setObject(1, teacher_id.getJdbc());
            ps.setObject(2, course_id.getJdbc());
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                teacher_id.setJdbc((Integer) rs.getObject(1));
                course_id.setJdbc((Integer) rs.getObject(2));
                name.setJdbc((String) rs.getObject(3));
                rs.close();
                ps.close();
                return 1;
            }
            else {
                rs.close();
                ps.close();
                return 0;
            }
        }
        catch (SQLException e) {
            throw e;
        }
    }
    public void doSelect(Connection con, String whereClause) throws
SQLException {
```

```
        String prepareString;
        try {
            if (whereClause.equals(""))
                prepareString = "select * from courses";
            else
                prepareString = "select * from courses where " +
whereClause;
            ps = con.prepareStatement(prepareString);
            rs = ps.executeQuery();
        }
        catch (SQLException e) {
            throw e;
        }
    }

    public int doNext() throws SQLException {
        try {
            if (rs.next()){
                teacher_id.setJdbc((Integer) rs.getObject(1));
                course_id.setJdbc((Integer) rs.getObject(2));
                name.setJdbc((String) rs.getObject(3));
                return 1;
            }
            else
                return 0;
        }
        catch (SQLException e) {
            throw(e);
        }
    }

    public void doClose() throws SQLException {
        try { rs.close(); ps.close(); }
        catch (SQLException e) { throw(e); }
    }
}
```

Db-tyypin perii luokka Courses. Siihen siis sovelluksen kirjoittaja voi kirjoittaa omaa koodiaan.

```
package fi.uta.cs.students;

public class Courses extends DbCourses{
    public Courses(){
        super();
    }
}
```

Courses-taulun SQL-luontilauseet esitetään tässä standardin SQL-kielen mukaisesti. Esimerkiksi MySQL-asetuksia käytettäessä drop table -lauseen perään lisättäisiin ”if exists” -mikä vähentää virheilmoituksia, jos lauseet ajetaan tyhjään kantaan, jossa drop table -lauseella poistettavia tauluja ei ole olemassa. Insert-lauseet saadaan taulun esimerkkitiedoista

```
drop table courses cascade;
create table courses(
teacher_id int,
course_id int,
name varchar(30),
primary key (teacher_id, course_id),
foreign key (teacher_id) references teachers(teacher_id));
insert into courses values (1, 1, 'Tulipallot I');
insert into courses values (3, 2, 'Maailman valtaaminen');
insert into courses values (3, 3, 'Hävitys');
insert into courses values (4, 4, 'Demonit II');
```

```
insert into courses values (5, 5, 'Telekinesia');
```

HTML-koodia relations.html-tiedostosta courses-taulun kohdalta. Taulukossa näytetään siis attribuutin nimi, mahdollinen null-rajoitus, yksilöivä attribuuttinimi, SQL-tietotyyppi, Java-tietotyyppi ja mahdollinen kuvaus. Yksilöivä attribuuttinimi tarkoittaa nimeä, joka ilman taulutietoakin yksilöi attribuutin kaikista muista tietokannan attribuuteista. Se luodaan lisäämällä taulun nimi attribuutin nimen eteen.

```
<p><p><a name='courses'></a> <b>COURSES</b>
<p><b>Description</b>:
<table><tr>
<td> <b>Relation attribute</b> </td>
<td> <b>Not Null</b> </td>
<td> <b>Base attribute Name</b> </td>
<td> <b>SQL Data Type</b> </td>
<td> <b>Java Data Type</b> </td>
<td> <b>Description</b> </td>
</tr>
<tr>
<td> teacher_id </td>
<td> false </td>
<td> courses_teacher_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
</tr>
<tr>
<td> course_id </td>
<td> false </td>
<td> courses_course_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
</tr>
<tr>
<td> name </td>
<td> false </td>
<td> courses_name </td>
<td> varchar(30) </td>
<td> SqlVarchar </td>
<td>  </td>
</tr>
</table>
primary key (  teacher_id, course_id  )
<br>foreign key (  teacher_id ) references teachers(teacher_id )
```

Kaikkien taulujen attribuutit listaava attributes.html listaa attribuuteista vähemmän ominaisuuksia kuin relations.html. Mukana on vain nimi, SQL-tietotyyppi, Java-tietotyyppi ja mahdollinen kuvaus.

```
<!doctype html public '-//w3c//dtd html 4.0
transitional//en'><html><head><meta http-equiv='Content-Type'
content='text/html; charset=iso-8859-
1'><title>Attributes</title></head><body text='#000000'
bgcolor='#FFFFFF' link='#0000EE' vlink='#551A8B'
alink='#FF0000'><table><tr>
<td> <b>Attribute Name</b> </td>
<td> <b>SQL Data Type</b> </td>
<td> <b>Java Data Type</b> </td>
<td> <b>Description</b> </td>
</tr>
<tr>
<td> <a name=' students_student_id '></a> students_student_id </td>
<td> int </td>
```

```
<td> SqlInteger </td>
<td> Student identification. </td>
</tr>
<tr>
<td> <a name=' students_name '></a> students_name </td>
<td> varchar(30) </td>
<td> SqlVarchar </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' students_major '></a> students_major </td>
<td> char(5) </td>
<td> SqlChar </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' teachers_teacher_id '></a> teachers_teacher_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' teachers_name '></a> teachers_name </td>
<td> varchar(30) </td>
<td> SqlVarchar </td>
<td> Name of
the teacher. </td>
</tr>
<tr>
<td> <a name=' teachers_room '></a> teachers_room </td>
<td> char(5) </td>
<td> SqlChar </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' teachers_sallary '></a> teachers_sallary </td>
<td> double(9,2) </td>
<td> SqlDecimal </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' teachers_start_date '></a> teachers_start_date </td>
<td> date </td>
<td> SqlDate </td>
<td> Employment began. </td>
</tr>
<tr>
<td> <a name=' courses_teacher_id '></a> courses_teacher_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' courses_course_id '></a> courses_course_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' courses_name '></a> courses_name </td>
<td> varchar(30) </td>
<td> SqlVarchar </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' grades_course_id '></a> grades_course_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
</tr>
```

```
<tr>
<td> <a name=' grades_student_id '></a> grades_student_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' grades_grade '></a> grades_grade </td>
<td> float(2,1) </td>
<td> SqlDecimal </td>
<td>  </td>
</tr>
</body>
 </html>
```

Liitteissä on esitetty luodut tiedostot kokonaisuudessaan.

9. Lopuksi

Olen tässä työssä kartoittanut aiempaa tietokantojen takaisinmallintamista koskevaa kirjallisuutta. Mukaan ottamani artikkelit ovat yhtä mieltä monista perusasioista, ja eroja löytyy lähinnä kunkin artikkelin uusista ideoista ja yksityiskohdissa.

Takaisinmallinnuksen kaksi suurta ongelmaa ovat eri attribuuttien merkitysten ja attribuuttien välisten suhteiden löytäminen. Menetelmät ja algoritmit vaativat erilaisia ennakko-oletuksia ja käyttäjävuorovaikutuskohtia ja antavat erilaisia tuloksia, ja tuottavat lisäksi eri määriä häiriötä ja jättävät huomaamatta piilotietoa. Juuri tämän vuoksi useat eri menetelmät ovat tarpeen.

Erilaiset ratkaisut takaisinmallinnusongelmiin voidaan yleisesti ottaen jakaa kahteen ryhmään. Yhtäältä löytyy erilaisia algoritmeja ja muita yksittäiseen ongelmaosaan keskittyviä menetelmiä ja toisaalta koko prosessin kulkuun päätyneitä mallinnuskäytäntöjä. Näitä pienempiä ratkaisuita voi yhdistää isompiin kehyksiin ja myös isompia kehysratkaisuita voi jossain tilanteissa yhdistellä. Esimerkiksi Jahnken et al. [1997] graafinen esitystapa sopii käyttöön mallinnettaessa oliokäytäntöjen mukaisesti.

Lopuksi tutkin takaisinmallinnusta kirjoittamalla Fujaba-ohjelmaan lisätoiminnallisuutta SQL-tietokantojen käsittelyyn, luomiseen ja takaisinmallintamiseen. Tietokantakaaviotoiminnallisuus pystyy tekemään kaavion SQL-lauseista tai tietokanta-ajurin antamien tietojen perusteella ja muokkauksen jälkeen tekemään muutokset tietokantaan käyttäjän ohjeistuksen perusteella ja luomaan tietokannasta HTML-esityksen, SQL-luontilauseet ja Java-koodin tietokannan käsittelyyn.

Viiteluettelo

[Blaha, 1997] Michael R. Blaha, Dimensions of database reverse engineering, *In: 4th Working Conference on Reverse Engineering (WCRE '97*), IEEE Computer Society, 1997, 176-183.

[Elmasri and Navathe, 2000] R. Elmasri and S. Navathe, *Fundamentals of Database Systems, 3rd ed.,* Addison-Wesley, 2000.

[Flach and Savnik, 1999] Peter A. Flach and Iztok Savnik, Database dependency discovery: A machine learning approach, *AI Communications* **12**, (3), 1999, 139-160.

[Hehrard and Hainaut, 2001] Jean Henrard and Jean-Luc Hainaut, Data dependency elicitation in database reverse engineering, In: *5th European Conference on Software Maintenance and Reengineering (CSMR 2001)*, IEEE Computer Society, 2001, 11-19.

[Henrard et al., 1999] Jean Henrard, Jean-Luc Hainaut, Jean-Marc Hick, Didier Roland and Vincent Englebert, Data structure extraction in database reverse engineering, In: Peter P. Chen, David W. Embley, Jacques Kouloumdjian, Stephen W. Liddle and John F. Roddick (eds.), *Advances in Conceptual Modeling: ER '99 Workshops on Evolution and Change in Data Management, Reverse Engineering in Information Systems, and the World Wide Web and Conceptual Modeling, Paris, France, November, 1999,* Springer, 149-160.

[Hehrard et al., 2002] Jean Henrard, Jean-Marc Hick, Philippe Thiran and Jean-Luc Hainaut, Strategies for Data Reengineering, In: *Proceedings of Ninth Working Conference on Reverse Engineering (WCRE'02)*, IEEE Computer Society Press, 2002, 211-220.

[Jahnke et al., 1997] Jens Jahnke, Wilhelm Schaefer and Albert Zuendorf, Generic Fuzzy Reasoning Nets as a Basis for Reverse Engineering Relational Database Applications, *In Proceedings of Sixth European Software Engineering Conference (ESEC '97)*, Springer, 1997, 193-210.

[Lim and Harrison, 1997] Wie Ming Lim and John Harrison, Discovery of constraints from data for information system reverse engineering, In: *Proceedings of Australian Software Engineering Conference, ASWEC '97*, IEEE Computer Society Press, 1997, 39-48.

[Mannila and Räihä, 1992] Heikki Mannila and Kari-Jouko Räihä, *The Design of Relational Databases*, Addison-Wesley, 1992.

[Nummenmaa, 1995] Jyrki Nummenmaa, *Designing Desirable Database Schemes*, Ph.D. Thesis, Department of Computer Science, University of Tampere, 1995.

[Nummenmaa and Seppi, 2004] Jyrki Nummenmaa and Ari Seppi, Automating SQL database evolution using a simple and intuitive maintenance operation model, *Submitted for publication,* 2004.

[Pedro-de-Jesus and Sousa, 1999] Maria de Lurdes Pedro-de-Jesus and Pedro Manuel Antunes Sousa, Selection of Reverse Engineering Methods for Relational Databases, In: *Proceedings of Third European Conference on Software Maintenance and Reengineering 1999*, IEEE Computer Society, 1999, 194-197.

[Premerlani and Blaha, 1992] William J. Premerlani and Michael R. Blaha, An approach for reverse engineering of relational databases, *Communications of the ACM* **37 (**5),1994, 42-49.

[Ramanathan and Hodges, 1997] Shekar Ramanathan and Julia E. Hodges, Extraction of object-oriented structures from existing relational databases, *SIGMOD Record* **26** (1), 1997, 59-64.

[Seppi and Nummenmaa, 2003] Ari Seppi and Jyrki Nummenmaa, A database schema diagram plugin for Fujaba, *Fujaba Days 2003*, 2003, 39-43.

[Soutou, 1996] Christian Soutou, Extracting N-ary Relationships Through Database Reverse Engineering, In: *Proceedings of International Conference on Conceptual Modeling 1996*, Springer, 1996, 392-405.

Liite 1: Java-koodit

DbCourses.java

```
package fi.uta.cs.students;

import java.io.PrintWriter;

import java.sql.*;

import fi.uta.cs.sqldatatypes.*;

public class DbCourses {
    private SqlInteger teacher_id;
    private SqlInteger teacher_id_temp;
    private SqlInteger course_id;
    private SqlInteger course_id_temp;
    private SqlVarchar name;
    private PreparedStatement ps;
    private ResultSet rs;

    public DbCourses() {
         teacher_id = new SqlInteger();
         teacher_id_temp = new SqlInteger();
         course_id = new SqlInteger();
         course_id_temp = new SqlInteger();
         name = new SqlVarchar(30, null);
         teacher_id.setPrime(true);
         course_id.setPrime(true);
    }

    public String toString() {
         StringBuffer sb = new StringBuffer();
         sb.append("courses\n");
         sb.append("teacher_id:" +  teacher_id.toString() + "\n");
         sb.append("course_id:" +  course_id.toString() + "\n");
         sb.append("name:" +  name.toString() + "\n");
         sb.append("\n");
         return(sb.toString());
    }

    public void output() {
         System.out.println("courses");
         System.out.println("teacher_id:" +  teacher_id.toString());
         System.out.println("course_id:" +  course_id.toString());
         System.out.println("name:" +  name.toString());
         System.out.println();
    }

    public void output(PrintWriter pw) {
         try { pw.write(toString()); }
         catch (Exception e) {System.out.println(e);}
    }

    public int setTeacher_id(String input) {
         return teacher_id.setAndCheck(input, false, "teacher_id");
    }

    public SqlInteger getTeacher_id() {
         return teacher_id;
    }


    public int setCourse_id(String input) {
         return course_id.setAndCheck(input, false, "course_id");
    }

    public SqlInteger getCourse_id() {
```

```
        return course_id;
    }


    public int setName(String input) {
        return name.setAndCheck(input, false, "name");
    }

    public SqlVarchar getName() {
        return name;
    }

    public int insert(Connection con) throws SQLException {
        try {
            String prepareString = "insert into  courses values
(?,?,?)";
            PreparedStatement ps = con.prepareStatement
(prepareString);
            ps.setObject(1, teacher_id.getJdbc());
            teacher_id_temp.setAndCheck(teacher_id.toString(), false,
"teacher_id_temp");
            ps.setObject(2, course_id.getJdbc());
            course_id_temp.setAndCheck(course_id.toString(), false,
"course_id_temp");
            ps.setObject(3, name.getJdbc());
            int rows = ps.executeUpdate();
            ps.close();
            return rows;
        }
        catch (SQLException e) {
            throw e;
        }
    }

    public int update(Connection con) throws SQLException {
        try {
            String prepareString = "update  courses" +
        " set teacher_id = ?, course_id = ?, name = ?" +
            " where teacher_id = ? AND course_id = ?;";
            PreparedStatement ps = con.prepareStatement
(prepareString);
            ps.setObject(1, teacher_id.getJdbc());
            ps.setObject(2, course_id.getJdbc());
            ps.setObject(3, name.getJdbc());
            ps.setObject(4, teacher_id_temp.get());
            ps.setObject(5, course_id_temp.get());
            int rows = ps.executeUpdate();
            ps.close();
            teacher_id_temp.setAndCheck(teacher_id.toString(), false,
"teacher_id_temp");
            course_id_temp.setAndCheck(course_id.toString(), false,
"course_id_temp");
            return rows;
        }
        catch (SQLException e) {
            throw e;
        }
    }

    public int delete(Connection con) throws SQLException {
        try {
            String prepareString = "delete from courses" +
            " where teacher_id = ? AND course_id = ?;";
            PreparedStatement ps = con.prepareStatement
(prepareString);
            ps.setObject(1, teacher_id.getJdbc());
            ps.setObject(2, course_id.getJdbc());
            int rows = ps.executeUpdate();
            ps.close();
            return rows;
```

```
        }
        catch (SQLException e) {
            throw e;
        }
    }

    public int select(Connection con) throws SQLException {
        try {
            String prepareString = "select * from  courses" +
            " where teacher_id = ? AND course_id = ?;";
            PreparedStatement ps = con.prepareStatement
(prepareString);
            ps.setObject(1, teacher_id.getJdbc());
            ps.setObject(2, course_id.getJdbc());
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                teacher_id.setJdbc((Integer) rs.getObject(1));
                course_id.setJdbc((Integer) rs.getObject(2));
                name.setJdbc((String) rs.getObject(3));
                rs.close();
                ps.close();
                return 1;
            }
            else {
                rs.close();
                ps.close();
                return 0;
            }
        }
        catch (SQLException e) {
            throw e;
        }
    }
    public void doSelect(Connection con, String whereClause) throws
SQLException {
        String prepareString;
        try {
            if (whereClause.equals(""))
                prepareString = "select * from courses";
            else
                prepareString = "select * from courses where " +
whereClause;
            ps = con.prepareStatement(prepareString);
            rs = ps.executeQuery();
        }
        catch (SQLException e) {
            throw e;
        }
    }

    public int doNext() throws SQLException {
        try {
            if (rs.next()){
                teacher_id.setJdbc((Integer) rs.getObject(1));
                course_id.setJdbc((Integer) rs.getObject(2));
                name.setJdbc((String) rs.getObject(3));
                return 1;
            }
            else
                return 0;
        }
        catch (SQLException e) {
            throw(e);
        }
    }

    public void doClose() throws SQLException {
        try { rs.close(); ps.close(); }
        catch (SQLException e) { throw(e); }
    }
```

```
}
```

Courses.java

```
package fi.uta.cs.students;

public class Courses extends DbCourses{
    public Courses(){
         super();
    }
}
```

DbGrades.java

```
package fi.uta.cs.students;

import java.io.PrintWriter;

import java.sql.*;

import fi.uta.cs.sqldatatypes.*;

public class DbGrades {
    private SqlInteger course_id;
    private SqlInteger course_id_temp;
    private SqlInteger student_id;
    private SqlInteger student_id_temp;
    private SqlDecimal grade;
    private PreparedStatement ps;
    private ResultSet rs;

    public DbGrades() {
         course_id = new SqlInteger();
         course_id_temp = new SqlInteger();
         student_id = new SqlInteger();
         student_id_temp = new SqlInteger();
         grade = new SqlDecimal(2, 1, null);
         course_id.setPrime(true);
         student_id.setPrime(true);
    }

    public String toString() {
         StringBuffer sb = new StringBuffer();
         sb.append("grades\n");
         sb.append("course_id:" +  course_id.toString() + "\n");
         sb.append("student_id:" +  student_id.toString() + "\n");
         sb.append("grade:" +  grade.toString() + "\n");
         sb.append("\n");
         return(sb.toString());
    }

    public void output() {
         System.out.println("grades");
         System.out.println("course_id:" +  course_id.toString());
         System.out.println("student_id:" +  student_id.toString());
         System.out.println("grade:" +  grade.toString());
         System.out.println();
    }

    public void output(PrintWriter pw) {
         try { pw.write(toString()); }
         catch (Exception e) {System.out.println(e);}
    }

    public int setCourse_id(String input) {
         return course_id.setAndCheck(input, false, "course_id");
    }

    public SqlInteger getCourse_id() {
         return course_id;
```

```
    }


    public int setStudent_id(String input) {
         return student_id.setAndCheck(input, false, "student_id");
    }

    public SqlInteger getStudent_id() {
         return student_id;
    }


    public int setGrade(String input) {
         return grade.setAndCheck(input, false, "grade");
    }

    public SqlDecimal getGrade() {
         return grade;
    }

    public int insert(Connection con) throws SQLException {
         try {
             String prepareString = "insert into  grades values(?,?,?)
";
             PreparedStatement ps = con.prepareStatement
(prepareString);
             ps.setObject(1, course_id.getJdbc());
             course_id_temp.setAndCheck(course_id.toString(), false,
"course_id_temp");
             ps.setObject(2, student_id.getJdbc());
             student_id_temp.setAndCheck(student_id.toString(), false,
"student_id_temp");
             ps.setObject(3, grade.getJdbc());
             int rows = ps.executeUpdate();
             ps.close();
             return rows;
         }
         catch (SQLException e) {
             throw e;
         }
    }

    public int update(Connection con) throws SQLException {
         try {
             String prepareString = "update  grades" +
         " set course_id = ?, student_id = ?, grade = ?" +
             " where course_id = ? AND student_id = ?;";
             PreparedStatement ps = con.prepareStatement
(prepareString);
             ps.setObject(1, course_id.getJdbc());
             ps.setObject(2, student_id.getJdbc());
             ps.setObject(3, grade.getJdbc());
             ps.setObject(4, course_id_temp.get());
             ps.setObject(5, student_id_temp.get());
             int rows = ps.executeUpdate();
             ps.close();
             course_id_temp.setAndCheck(course_id.toString(), false,
"course_id_temp");
             student_id_temp.setAndCheck(student_id.toString(), false,
"student_id_temp");
             return rows;
         }
         catch (SQLException e) {
             throw e;
         }
    }

    public int delete(Connection con) throws SQLException {
         try {
             String prepareString = "delete from grades" +
```

```
            " where course_id = ? AND student_id = ?;";
            PreparedStatement ps = con.prepareStatement
(prepareString);
            ps.setObject(1, course_id.getJdbc());
            ps.setObject(2, student_id.getJdbc());
            int rows = ps.executeUpdate();
            ps.close();
            return rows;
        }
        catch (SQLException e) {
            throw e;
        }
    }

    public int select(Connection con) throws SQLException {
        try {
            String prepareString = "select * from  grades" +
            " where course_id = ? AND student_id = ?;";
            PreparedStatement ps = con.prepareStatement
(prepareString);
            ps.setObject(1, course_id.getJdbc());
            ps.setObject(2, student_id.getJdbc());
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                course_id.setJdbc((Integer) rs.getObject(1));
                student_id.setJdbc((Integer) rs.getObject(2));
                grade.setJdbc((Decimal) rs.getObject(3));
                rs.close();
                ps.close();
                return 1;
            }
            else {
                rs.close();
                ps.close();
                return 0;
            }
        }
        catch (SQLException e) {
            throw e;
        }
    }
    public void doSelect(Connection con, String whereClause) throws
SQLException {
        String prepareString;
        try {
            if (whereClause.equals(""))
                prepareString = "select * from grades";
            else
                prepareString = "select * from grades where " +
whereClause;
            ps = con.prepareStatement(prepareString);
            rs = ps.executeQuery();
        }
        catch (SQLException e) {
            throw e;
        }
    }

    public int doNext() throws SQLException {
        try {
            if (rs.next()){
                course_id.setJdbc((Integer) rs.getObject(1));
                student_id.setJdbc((Integer) rs.getObject(2));
                grade.setJdbc((Decimal) rs.getObject(3));
                return 1;
            }
            else
                return 0;
        }
        catch (SQLException e) {
```

```
            throw(e);
        }
    }

    public void doClose() throws SQLException {
        try { rs.close(); ps.close(); }
        catch (SQLException e) { throw(e); }
    }
}
```

Grades.java

```
package fi.uta.cs.students;

public class Grades extends DbGrades{
    public Grades(){
        super();
    }
}
```

DbStudents.java

```
package fi.uta.cs.students;

import java.io.PrintWriter;

import java.sql.*;

import fi.uta.cs.sqldatatypes.*;

// This is a student table.

public class DbStudents {
    private SqlInteger student_id;
    private SqlInteger student_id_temp;
    private SqlVarchar name;
    private SqlChar major;
    private PreparedStatement ps;
    private ResultSet rs;

    public DbStudents() {
    // Student identification.
        student_id = new SqlInteger();
        student_id_temp = new SqlInteger();
        name = new SqlVarchar(30, null);
        major = new SqlChar(5, null);
        student_id.setPrime(true);
    }

    public String toString() {
        StringBuffer sb = new StringBuffer();
        sb.append("students\n");
        sb.append("student_id:" +  student_id.toString() + "\n");
        sb.append("name:" +  name.toString() + "\n");
        sb.append("major:" +  major.toString() + "\n");
        sb.append("\n");
        return(sb.toString());
    }

    public void output() {
        System.out.println("students");
        System.out.println("student_id:" +  student_id.toString());
        System.out.println("name:" +  name.toString());
        System.out.println("major:" +  major.toString());
        System.out.println();
    }

    public void output(PrintWriter pw) {
        try { pw.write(toString()); }
        catch (Exception e) {System.out.println(e);}
```

```
    }

    public int setStudent_id(String input) {
         return student_id.setAndCheck(input, false, "student_id");
    }

    public SqlInteger getStudent_id() {
         return student_id;
    }


    public int setName(String input) {
         return name.setAndCheck(input, false, "name");
    }

    public SqlVarchar getName() {
         return name;
    }


    public int setMajor(String input) {
         return major.setAndCheck(input, false, "major");
    }

    public SqlChar getMajor() {
         return major;
    }

    public int insert(Connection con) throws SQLException {
         try {
             String prepareString = "insert into  students values
(?,?,?)";
             PreparedStatement ps = con.prepareStatement
(prepareString);
             ps.setObject(1, student_id.getJdbc());
             student_id_temp.setAndCheck(student_id.toString(), false,
"student_id_temp");
             ps.setObject(2, name.getJdbc());
             ps.setObject(3, major.getJdbc());
             int rows = ps.executeUpdate();
             ps.close();
             return rows;
         }
         catch (SQLException e) {
             throw e;
         }
    }

    public int update(Connection con) throws SQLException {
         try {
             String prepareString = "update  students" +
         " set student_id = ?, name = ?, major = ?" +
             " where student_id = ?;";
             PreparedStatement ps = con.prepareStatement
(prepareString);
             ps.setObject(1, student_id.getJdbc());
             ps.setObject(2, name.getJdbc());
             ps.setObject(3, major.getJdbc());
             ps.setObject(4, student_id_temp.get());
             int rows = ps.executeUpdate();
             ps.close();
             student_id_temp.setAndCheck(student_id.toString(), false,
"student_id_temp");
             return rows;
         }
         catch (SQLException e) {
             throw e;
         }
    }
```

```
    public int delete(Connection con) throws SQLException {
         try {
             String prepareString = "delete from students" +
             " where student_id = ?;";
             PreparedStatement ps = con.prepareStatement
(prepareString);
             ps.setObject(1, student_id.getJdbc());
             int rows = ps.executeUpdate();
             ps.close();
             return rows;
         }
         catch (SQLException e) {
             throw e;
         }
    }

    public int select(Connection con) throws SQLException {
         try {
             String prepareString = "select * from  students" +
             " where student_id = ?;";
             PreparedStatement ps = con.prepareStatement
(prepareString);
             ps.setObject(1, student_id.getJdbc());
             ResultSet rs = ps.executeQuery();
             if (rs.next()) {
                 student_id.setJdbc((Integer) rs.getObject(1));
                 name.setJdbc((String) rs.getObject(2));
                 major.setJdbc((String) rs.getObject(3));
                 rs.close();
                 ps.close();
                 return 1;
             }
             else {
                 rs.close();
                 ps.close();
                 return 0;
             }
         }
         catch (SQLException e) {
             throw e;
         }
    }
    public void doSelect(Connection con, String whereClause) throws
SQLException {
         String prepareString;
         try {
             if (whereClause.equals(""))
                 prepareString = "select * from students";
             else
                 prepareString = "select * from students where " +
whereClause;
             ps = con.prepareStatement(prepareString);
             rs = ps.executeQuery();
         }
         catch (SQLException e) {
             throw e;
         }
    }

    public int doNext() throws SQLException {
         try {
             if (rs.next()){
                 student_id.setJdbc((Integer) rs.getObject(1));
                 name.setJdbc((String) rs.getObject(2));
                 major.setJdbc((String) rs.getObject(3));
                 return 1;
             }
             else
                 return 0;
         }
```

```
          catch (SQLException e) {
              throw(e);
          }
     }

     public void doClose() throws SQLException {
          try { rs.close(); ps.close(); }
          catch (SQLException e) { throw(e); }
     }
}
```

Studens.java

```
package fi.uta.cs.students;

public class Students extends DbStudents{
     public Students(){
          super();
     }
}
```

DbTeachers.java

```
package fi.uta.cs.students;

import java.io.PrintWriter;

import java.sql.*;

import fi.uta.cs.sqldatatypes.*;

public class DbTeachers {
     private SqlInteger teacher_id;
     private SqlInteger teacher_id_temp;
     private SqlVarchar name;
     private SqlChar room;
     private SqlDecimal sallary;
     private SqlDate start_date;
     private PreparedStatement ps;
     private ResultSet rs;

     public DbTeachers() {
          teacher_id = new SqlInteger();
          teacher_id_temp = new SqlInteger();
     // Name of
     // the teacher.
          name = new SqlVarchar(30, null);
          room = new SqlChar(5, null);
          sallary = new SqlDecimal(9, 2, null);
     // Employment began.
          start_date = new SqlDate();
          teacher_id.setPrime(true);
     }

     public String toString() {
          StringBuffer sb = new StringBuffer();
          sb.append("teachers\n");
          sb.append("teacher_id:" +  teacher_id.toString() + "\n");
          sb.append("name:" +  name.toString() + "\n");
          sb.append("room:" +  room.toString() + "\n");
          sb.append("sallary:" +  sallary.toString() + "\n");
          sb.append("start_date:" +  start_date.toString() + "\n");
          sb.append("\n");
          return(sb.toString());
     }

     public void output() {
          System.out.println("teachers");
          System.out.println("teacher_id:" +  teacher_id.toString());
          System.out.println("name:" +  name.toString());
```

```
          System.out.println("room:" +  room.toString());
          System.out.println("sallary:" +  sallary.toString());
          System.out.println("start_date:" +  start_date.toString());
          System.out.println();
     }

     public void output(PrintWriter pw) {
          try { pw.write(toString()); }
          catch (Exception e) {System.out.println(e);}
     }

     public int setTeacher_id(String input) {
          return teacher_id.setAndCheck(input, false, "teacher_id");
     }

     public SqlInteger getTeacher_id() {
          return teacher_id;
     }


     public int setName(String input) {
          return name.setAndCheck(input, false, "name");
     }

     public SqlVarchar getName() {
          return name;
     }


     public int setRoom(String input) {
          return room.setAndCheck(input, false, "room");
     }

     public SqlChar getRoom() {
          return room;
     }


     public int setSallary(String input) {
          return sallary.setAndCheck(input, false, "sallary");
     }

     public SqlDecimal getSallary() {
          return sallary;
     }


     public int setStart_date(String input) {
          return start_date.setAndCheck(input, false, "start_date");
     }

     public SqlDate getStart_date() {
          return start_date;
     }

     public int insert(Connection con) throws SQLException {
          try {
               String prepareString = "insert into  teachers values
(?,?,?,?,?)";
               PreparedStatement ps = con.prepareStatement
(prepareString);
               ps.setObject(1, teacher_id.getJdbc());
               teacher_id_temp.setAndCheck(teacher_id.toString(), false,
"teacher_id_temp");
               ps.setObject(2, name.getJdbc());
               ps.setObject(3, room.getJdbc());
               ps.setObject(4, sallary.getJdbc());
               ps.setObject(5, start_date.getJdbc());
               int rows = ps.executeUpdate();
               ps.close();
```

```
            return rows;
        }
        catch (SQLException e) {
            throw e;
        }
    }

    public int update(Connection con) throws SQLException {
        try {
            String prepareString = "update  teachers" +
        " set teacher_id = ?, name = ?, room = ?, sallary = ?,
start_date = ?" +
            " where teacher_id = ?;";
            PreparedStatement ps = con.prepareStatement
(prepareString);
            ps.setObject(1, teacher_id.getJdbc());
            ps.setObject(2, name.getJdbc());
            ps.setObject(3, room.getJdbc());
            ps.setObject(4, sallary.getJdbc());
            ps.setObject(5, start_date.getJdbc());
            ps.setObject(6, teacher_id_temp.get());
            int rows = ps.executeUpdate();
            ps.close();
            teacher_id_temp.setAndCheck(teacher_id.toString(), false,
"teacher_id_temp");
            return rows;
        }
        catch (SQLException e) {
            throw e;
        }
    }

    public int delete(Connection con) throws SQLException {
        try {
            String prepareString = "delete from teachers" +
            " where teacher_id = ?;";
            PreparedStatement ps = con.prepareStatement
(prepareString);
            ps.setObject(1, teacher_id.getJdbc());
            int rows = ps.executeUpdate();
            ps.close();
            return rows;
        }
        catch (SQLException e) {
            throw e;
        }
    }

    public int select(Connection con) throws SQLException {
        try {
            String prepareString = "select * from  teachers" +
            " where teacher_id = ?;";
            PreparedStatement ps = con.prepareStatement
(prepareString);
            ps.setObject(1, teacher_id.getJdbc());
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                teacher_id.setJdbc((Integer) rs.getObject(1));
                name.setJdbc((String) rs.getObject(2));
                room.setJdbc((String) rs.getObject(3));
                sallary.setJdbc((Decimal) rs.getObject(4));
                start_date.setJdbc((Date) rs.getObject(5));
                rs.close();
                ps.close();
                return 1;
            }
            else {
                rs.close();
                ps.close();
                return 0;
```

```
            }
        }
        catch (SQLException e) {
            throw e;
        }
    }
    public void doSelect(Connection con, String whereClause) throws
SQLException {
        String prepareString;
        try {
            if (whereClause.equals(""))
                prepareString = "select * from teachers";
            else
                prepareString = "select * from teachers where " +
whereClause;
            ps = con.prepareStatement(prepareString);
            rs = ps.executeQuery();
        }
        catch (SQLException e) {
            throw e;
        }
    }

    public int doNext() throws SQLException {
        try {
            if (rs.next()){
                teacher_id.setJdbc((Integer) rs.getObject(1));
                name.setJdbc((String) rs.getObject(2));
                room.setJdbc((String) rs.getObject(3));
                sallary.setJdbc((Decimal) rs.getObject(4));
                start_date.setJdbc((Date) rs.getObject(5));
                return 1;
            }
            else
                return 0;
        }
        catch (SQLException e) {
            throw(e);
        }
    }

    public void doClose() throws SQLException {
        try { rs.close(); ps.close(); }
        catch (SQLException e) { throw(e); }
    }
}
```

Teachers.java

```
package fi.uta.cs.students;

public class Teachers extends DbTeachers{
    public Teachers(){
        super();
    }
}
```

DbTeacher_courses.java

```
package fi.uta.cs.students;

import java.sql.*;

import fi.uta.cs.sqldatatypes.*;

// Kurssiopet.
// On tässä.

public class DbTeacher_courses {
    // Name of
```

```
    // the teacher.
    private SqlVarchar teacher_name;
    private SqlVarchar course_name;
    private PreparedStatement ps;
    private ResultSet rs;

    public DbTeacher_courses() {
         teacher_name = new SqlVarchar(30, null);
         course_name = new SqlVarchar(30, null);
    }

    public void output() {
         System.out.println("teacher_courses");
         System.out.println("teacher_name:" +  teacher_name.toString
());
         System.out.println("course_name:" +  course_name.toString());
         System.out.println();
    }

    public int setTeacher_name(String input) {
         return teacher_name.setAndCheck(input, false,
"teacher_name");
    }

    public SqlVarchar getTeacher_name() {
         return teacher_name;
    }


    public int setCourse_name(String input) {
         return course_name.setAndCheck(input, false, "course_name");
    }

    public SqlVarchar getCourse_name() {
         return course_name;
    }

    public int insert(Connection con) throws SQLException {
         try {
             String prepareString = "insert into  teacher_courses
values(?,?)";
             PreparedStatement ps = con.prepareStatement
(prepareString);
             ps.setObject(1, teacher_name.getJdbc());
             ps.setObject(2, course_name.getJdbc());
             int rows = ps.executeUpdate();
             ps.close();
             return rows;
         }
         catch (SQLException e) {
             throw e;
         }
    }

    public int delete(Connection con) throws SQLException {
         try {
             String prepareString = "delete from teacher_courses" +
             " where teacher_name = ?, course_name = ?;";
             PreparedStatement ps = con.prepareStatement
(prepareString);
             ps.setObject(0, teacher_name.getJdbc());
             ps.setObject(1, course_name.getJdbc());
             int rows = ps.executeUpdate();
             ps.close();
             return rows;
         }
         catch (SQLException e) {
             throw e;
         }
    }
```

```
    public int select(Connection con) throws SQLException {
         try {
             String prepareString = "select * from  teacher_courses" +
             " where teacher_name = ? AND course_name = ?;";
             PreparedStatement ps = con.prepareStatement
(prepareString);
             ps.setObject(0, teacher_name.getJdbc());
             ps.setObject(1, course_name.getJdbc());
             ResultSet rs = ps.executeQuery();
             if (rs.next()) {
                 teacher_name.setJdbc((String) rs.getObject(1));
                 course_name.setJdbc((String) rs.getObject(2));
                 rs.close();
                 ps.close();
                 return 1;
             }
             else {
                 rs.close();
                 ps.close();
                 return 0;
             }
         }
         catch (SQLException e) {
             throw e;
         }
    }

    public void doSelect(Connection con, String whereClause) throws
SQLException {
         String prepareString;
         try {
             if (whereClause.equals(""))
                 prepareString = "select * from teacher_courses";
             else
                 prepareString = "select * from teacher_courses where
" + whereClause;
             ps = con.prepareStatement(prepareString);
             rs = ps.executeQuery();
         }
         catch (SQLException e) {
             throw e;
         }
    }

    public int doNext() throws SQLException {
         try {
             if (rs.next()){
                 teacher_name.setJdbc((String) rs.getObject(1));
                 course_name.setJdbc((String) rs.getObject(2));
                 return 1;
             }
             else
                 return 0;
         }
         catch (SQLException e) {
             throw(e);
         }
    }

    public void doClose() throws SQLException {
         try { rs.close(); ps.close(); }
         catch (SQLException e) { throw(e); }
    }
}
```

Teacher_courses.java

```
package fi.uta.cs.students;
```

```
public class Teacher_courses extends DbTeacher_courses{
    public Teacher_courses(){
         super();
    }
}
```

DbTeachers_students.java

```
package fi.uta.cs.students;

import java.sql.*;

import fi.uta.cs.sqldatatypes.*;
import fi.uta.cs.students.*;

public class DbTeachers_students {
    private Teachers teachers;
    private Courses courses;
    private Grades grades;
    private Students students;
    private PreparedStatement ps;
    private ResultSet rs;

    public DbTeachers_students() {
         teachers = new Teachers();
         courses = new Courses();
         grades = new Grades();
         students = new Students();
    }


    public Teachers getTeachers(){
         return teachers;
    }


    public Courses getCourses(){
         return courses;
    }


    public Grades getGrades(){
         return grades;
    }


    public Students getStudents(){
         return students;
    }

    public void doSelect(Connection con, String whereClause) throws
SQLException {
         String prepareString;
         try {
             if (whereClause.equals(""))
                 prepareString = "select * from  teachers, courses,
grades, students " +
                 " where (teachers.teacher_id = courses.teacher_id)
AND (courses.course_id = grades.course_id) AND (grades.student_id =
students.student_id)";
             else
                 prepareString = "select * from  teachers, courses,
grades, students" +
                  " where ( (teachers.teacher_id = courses.teacher_id)
AND (courses.course_id = grades.course_id) AND (grades.student_id =
students.student_id) ) and " + whereClause;
             ps = con.prepareStatement(prepareString);
             rs = ps.executeQuery();
         }
         catch (SQLException e) {
```

```
            throw e;
        }
    }

    public int doNext() throws SQLException {
        try {
            if (rs.next()){
                teachers.getTeacher_id().setJdbc((Integer)
rs.getObject(1));
                teachers.getName().setJdbc((String) rs.getObject(2));
                teachers.getRoom().setJdbc((String) rs.getObject(3));
                teachers.getSallary().setJdbc((Decimal) rs.getObject
(4));
                teachers.getStart_date().setJdbc((Date) rs.getObject
(5));
                courses.getTeacher_id().setJdbc((Integer)
rs.getObject(6));
                courses.getCourse_id().setJdbc((Integer) rs.getObject
(7));
                courses.getName().setJdbc((String) rs.getObject(8));
                grades.getCourse_id().setJdbc((Integer) rs.getObject
(9));
                grades.getStudent_id().setJdbc((Integer) rs.getObject
(10));
                grades.getGrade().setJdbc((Decimal) rs.getObject
(11));
                students.getStudent_id().setJdbc((Integer)
rs.getObject(12));
                students.getName().setJdbc((String) rs.getObject
(13));
                students.getMajor().setJdbc((String) rs.getObject
(14));
                return 1;
            }
            else
                return 0;
        }
        catch (SQLException e) {
            throw(e);
        }
    }

    public void doClose() throws SQLException {
        try { rs.close(); ps.close(); }
        catch (SQLException e) { throw(e); }
    }
}
```

Teachers_students.java

```
package fi.uta.cs.students;

public class Teachers_students extends DbTeachers_students{
    public Teachers_students(){
        super();
    }
}
```

Liite 2: SQL-lauseet

all.sql

```
drop table students cascade;
create table students(
student_id int,
name varchar(30),
major char(5),
primary key (student_id));
insert into students values (1, 'Harry Potter', 'MAG');
insert into students values (2, 'Frodo', 'EAT');
insert into students values (3, 'Sam', 'EAT');
insert into students values (4, 'Buffy', 'PE');
insert into students values (5, 'Hermione Granger', 'MAG');
drop table teachers cascade;
create table teachers(
teacher_id int,
name varchar(30),
room char(5),
sallary double(9,2),
start_date date,
primary key (teacher_id));
insert into teachers values (1, 'Gandalf', 'A102', 2300.45, null);
insert into teachers values (2, 'Saruman', 'A340', 4534.55, null);
insert into teachers values (3, 'Sauron', 'A220', 5432.33, null);
insert into teachers values (4, 'Giles', 'B439', 4554.45, null);
insert into teachers values (5, 'Dubledore', 'C123', 5432.43, null);
drop table courses cascade;
create table courses(
teacher_id int,
course_id int,
name varchar(30),
primary key (teacher_id, course_id),
foreign key (teacher_id) references teachers(teacher_id));
insert into courses values (1, 1, 'Tulipallot I');
insert into courses values (3, 2, 'Maailman valtaaminen');
insert into courses values (3, 3, 'Hävitys');
insert into courses values (4, 4, 'Demonit II');
insert into courses values (5, 5, 'Telekinesia');
drop table grades cascade;
create table grades(
course_id int,
student_id int,
grade float(2,1),
primary key (course_id, student_id),
foreign key (course_id) references courses(course_id),
foreign key (student_id) references students(student_id));
insert into grades values (4, 4, 2.3);
insert into grades values (1, 2, 1.6);
insert into grades values (5, 4, 1.3);
insert into grades values (5, 5, 2.6);
insert into grades values (3, 5, 2.3);
```

Liite 3: HTML-tiedostot

relations.html

```
<!doctype html public '-//w3c//dtd html 4.0
transitional//en'><html><head><meta http-equiv='Content-Type'
content='text/html; charset=iso-8859-
1'><title>Relations</title></head><body text='#000000'
bgcolor='#FFFFFF' link='#0000EE' vlink='#551A8B'
alink='#FF0000'><table><tr>
<td> <b>Relation name</b> </td>
</tr>
<tr>
<td> <a href='#students'> STUDENTS </a></td>
</tr>
<tr>
<td> <a href='#teachers'> TEACHERS </a></td>
</tr>
<tr>
<td> <a href='#courses'> COURSES </a></td>
</tr>
<tr>
<td> <a href='#grades'> GRADES </a></td>
</tr>
</table><p><p><p><a name='students'></a> <b>STUDENTS</b>
<p><b>Description</b>: This is a student table.
<table><tr>
<td> <b>Relation attribute</b> </td>
<td> <b>Not Null</b> </td>
<td> <b>Base attribute Name</b> </td>
<td> <b>SQL Data Type</b> </td>
<td> <b>Java Data Type</b> </td>
<td> <b>Description</b> </td>
</tr>
<tr>
<td> student_id </td>
<td> false </td>
<td> students_student_id </td>
<td> int </td>
<td> SqlInteger </td>
<td> Student identification. </td>
</tr>
<tr>
<td> name </td>
<td> false </td>
<td> students_name </td>
<td> varchar(30) </td>
<td> SqlVarchar </td>
<td>  </td>
</tr>
<tr>
<td> major </td>
<td> false </td>
<td> students_major </td>
<td> char(5) </td>
<td> SqlChar </td>
<td>  </td>
</tr>
</table>
primary key (  student_id  )
<p><p><a name='teachers'></a> <b>TEACHERS</b>
<p><b>Description</b>:
<table><tr>
<td> <b>Relation attribute</b> </td>
<td> <b>Not Null</b> </td>
<td> <b>Base attribute Name</b> </td>
<td> <b>SQL Data Type</b> </td>
<td> <b>Java Data Type</b> </td>
```

```
<td> <b>Description</b> </td>
</tr>
<tr>
<td> teacher_id </td>
<td> false </td>
<td> teachers_teacher_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
</tr>
<tr>
<td> name </td>
<td> false </td>
<td> teachers_name </td>
<td> varchar(30) </td>
<td> SqlVarchar </td>
<td> Name of
the teacher. </td>
</tr>
<tr>
<td> room </td>
<td> false </td>
<td> teachers_room </td>
<td> char(5) </td>
<td> SqlChar </td>
<td>  </td>
</tr>
<tr>
<td> sallary </td>
<td> false </td>
<td> teachers_sallary </td>
<td> double(9,2) </td>
<td> SqlDecimal </td>
<td>  </td>
</tr>
<tr>
<td> start_date </td>
<td> false </td>
<td> teachers_start_date </td>
<td> date </td>
<td> SqlDate </td>
<td> Employment began. </td>
</tr>
</table>
primary key (  teacher_id  )
<p><p><a name='courses'></a> <b>COURSES</b>
<p><b>Description</b>:
<table><tr>
<td> <b>Relation attribute</b> </td>
<td> <b>Not Null</b> </td>
<td> <b>Base attribute Name</b> </td>
<td> <b>SQL Data Type</b> </td>
<td> <b>Java Data Type</b> </td>
<td> <b>Description</b> </td>
</tr>
<tr>
<td> teacher_id </td>
<td> false </td>
<td> courses_teacher_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
</tr>
<tr>
<td> course_id </td>
<td> false </td>
<td> courses_course_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
```

```
</tr>
<tr>
<td> name </td>
<td> false </td>
<td> courses_name </td>
<td> varchar(30) </td>
<td> SqlVarchar </td>
<td>  </td>
</tr>
</table>
primary key (  teacher_id, course_id  )
<br>foreign key (  teacher_id ) references teachers(teacher_id )
<p><p><a name='grades'></a> <b>GRADES</b>
<p><b>Description</b>:
<table><tr>
<td> <b>Relation attribute</b> </td>
<td> <b>Not Null</b> </td>
<td> <b>Base attribute Name</b> </td>
<td> <b>SQL Data Type</b> </td>
<td> <b>Java Data Type</b> </td>
<td> <b>Description</b> </td>
</tr>
<tr>
<td> course_id </td>
<td> false </td>
<td> grades_course_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
</tr>
<tr>
<td> student_id </td>
<td> false </td>
<td> grades_student_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
</tr>
<tr>
<td> grade </td>
<td> false </td>
<td> grades_grade </td>
<td> float(2,1) </td>
<td> SqlDecimal </td>
<td>  </td>
</tr>
</table>
primary key (  course_id, student_id  )
<br>foreign key (  course_id ) references courses(course_id )
<br>foreign key (  student_id ) references students(student_id )
</body>
 </html>
```

views.html

```
<!doctype html public '-//w3c//dtd html 4.0
transitional//en'><html><head><meta http-equiv='Content-Type'
content='text/html; charset=iso-8859-
1'><title>Queries</title></head><body text='#000000' bgcolor='#FFFFFF'
link='#0000EE' vlink='#551A8B' alink='#FF0000'><table><tr>
<td> <b>View Name</b> </td>
</tr>
<tr>
<td> <a href='#teacher_courses'> TEACHER_COURSES </a></td>
</tr>
</table><p><p><p><a name='teacher_courses'></a> <b>TEACHER_COURSES</b>
<p><b>Description</b>: Kurssiopet.
On tässä.
<table><tr>
<td> <b>View Attribute</b> </td>
```

```
<td> <b>Base attribute Name</b> </td>
<td> <b>SQL Data Type</b> </td>
<td> <b>Java Data Type</b> </td>
<td> <b>Original Attribute Name</b> </td>
<td> <b>Original Table Name</b> </td>
<td> <b>Description</b> </td>
</tr>
<tr>
<td> teacher_name </td>
<td> teachers_name </td>
<td> varchar(30) </td>
<td> SqlVarchar </td>
<td> name </td>
<td> teachers </td>
<td> Name of
the teacher. </td>
</tr>
<tr>
<td> course_name </td>
<td> courses_name </td>
<td> varchar(30) </td>
<td> SqlVarchar </td>
<td> name </td>
<td> courses </td>
<td>  </td>
</tr>
</table>
</body>
 </html>
```

queries.html

```
<!doctype html public '-//w3c//dtd html 4.0
transitional//en'><html><head><meta http-equiv='Content-Type'
content='text/html; charset=iso-8859-
1'><title>Queries</title></head><body text='#000000' bgcolor='#FFFFFF'
link='#0000EE' vlink='#551A8B' alink='#FF0000'><table><tr>
<td> <b>Query name</b> </td>
</tr>
<tr>
<td> <a href='#teachers_students'> TEACHERS_STUDENTS </a></td>
</tr>
</table><p><p><p>
<a name='teachers_students'></a> <b>TEACHERS_STUDENTS</b><br>
select * from teachers, courses, grades, students<br>
 where (teachers.teacher_id = courses.teacher_id) AND
(courses.course_id = grades.course_id) AND (grades.student_id =
students.student_id)<br>
</body>
 </html>
```

attributes.html

```
<!doctype html public '-//w3c//dtd html 4.0
transitional//en'><html><head><meta http-equiv='Content-Type'
content='text/html; charset=iso-8859-
1'><title>Attributes</title></head><body text='#000000'
bgcolor='#FFFFFF' link='#0000EE' vlink='#551A8B'
alink='#FF0000'><table><tr>
<td> <b>Attribute Name</b> </td>
<td> <b>SQL Data Type</b> </td>
<td> <b>Java Data Type</b> </td>
<td> <b>Description</b> </td>
</tr>
<tr>
<td> <a name=' students_student_id '></a> students_student_id </td>
<td> int </td>
<td> SqlInteger </td>
<td> Student identification. </td>
</tr>
```

```
<tr>
<td> <a name=' students_name '></a> students_name </td>
<td> varchar(30) </td>
<td> SqlVarchar </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' students_major '></a> students_major </td>
<td> char(5) </td>
<td> SqlChar </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' teachers_teacher_id '></a> teachers_teacher_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' teachers_name '></a> teachers_name </td>
<td> varchar(30) </td>
<td> SqlVarchar </td>
<td> Name of
the teacher. </td>
</tr>
<tr>
<td> <a name=' teachers_room '></a> teachers_room </td>
<td> char(5) </td>
<td> SqlChar </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' teachers_sallary '></a> teachers_sallary </td>
<td> double(9,2) </td>
<td> SqlDecimal </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' teachers_start_date '></a> teachers_start_date </td>
<td> date </td>
<td> SqlDate </td>
<td> Employment began. </td>
</tr>
<tr>
<td> <a name=' courses_teacher_id '></a> courses_teacher_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' courses_course_id '></a> courses_course_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' courses_name '></a> courses_name </td>
<td> varchar(30) </td>
<td> SqlVarchar </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' grades_course_id '></a> grades_course_id </td>
<td> int </td>
<td> SqlInteger </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' grades_student_id '></a> grades_student_id </td>
<td> int </td>
```

```
<td> SqlInteger </td>
<td>  </td>
</tr>
<tr>
<td> <a name=' grades_grade '></a> grades_grade </td>
<td> float(2,1) </td>
<td> SqlDecimal </td>
<td>  </td>
</tr>
</body>
 </html>
```