Teo Laitinen

POIKKEAMAPERUSTEINEN VÄÄRINKÄYTÖNTUNNISTAMINEN RELAATIOTIETOKANNANHALLINTAJÄRJESTELMISSÄ

Tietojärjestelmätieteen
pro gradu -tutkielma
30.9.2002

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Laitinen, Teo Markus
Poikkeamaperusteinen väärinkäytöntunnistaminen relaatiotietokannanhallintajärjestelmissä / Teo Laitinen
Jyväskylä: Jyväskylän yliopisto, 2002.
146 s.
Tutkielma

Tässä tutkielmassa tarkastellaan väärinkäytön automaattista tunnistamista relaatiotietokannanhallintajärjestelmien kontekstissa. Tutkielmassa on analysoitu aiempaa yleistä ja erityisesti tietokantajärjestelmiä koskevaa tutkimusta sekä jäsennetty tutkimusalan käsitteistöä. Yleisestä tutkimuksesta on haettu tietokannanhallintajärjestelmien yhteyteen sopivia malleja, joiden parhaita ominaisuuksia on tarkasteltu yksityiskohtaisemmin. Tietokannoille spesifistä tutkimuksesta on puolestaan etsitty tapoja, joilla normaalitoimintaa ja -käyttäytymistä voidaan mallintaa ja tunnistaa riittävällä tarkkuudella.

Tietokantajärjestelmässä tärkeimmäksi osa-alueeksi ja haastavimmaksi tehtäväksi osoittautui jäljitystietojen kerääminen, johon poikkeamia tunnistava järjestelmä perustuu sekä opetus- että tunnistusvaiheessa. Aiemman tutkimuksen jättämien epäselvien kysymysten tutkimiseksi rakennettiin prototyyppijärjestelmä, jolla jäljitystietojen keräämistä ja sen ongelmia pystyttiin selvittämään. Prototyyppi sisältää osat myös normaalitoiminnan ja -käyttäytymisen profiilien muodostamiseksi sekä poikkeamien tunnistamiseksi, mutta sen monipuolisin ominaisuus on jäljitystietojen hankinta.

Tutkielman keskeisiä tuloksia ovat poikkeamien tunnistuksen tarkempi käsitteellinen jako, joka erottaa käyttäjän toiminnan ja käyttäytymisen omiksi tunnistettaviksi kohteikseen sekä tämän perusajatuksen mukaan toteutettu prototyyppi. Yhtä lailla merkittävä tulos on prototyypissä toteutettu monipuolinen jäljitystietojen hankintajärjestelmä.

AVAINSANAT: tunkeutujan tunnistaminen, väärinkäytön tunnistaminen, poikkeamien tunnistaminen, tietokannat, tietoturva

SISÄLLYS

1 JOHDANTO

Yritysten ja organisaatioiden tarve kytkeä tietojärjestelmänsä julkisiin verkkoihin sekä luoda sähköisen kaupankäynnin ja asioinnin järjestelmiä, jotka ovat reaaliaikaisia, avoimia ja yhä monipuolisempia, asettaa tietoturvalle entistä kovemmat vaatimukset. Uusia aktiivisia ja helppokäyttöisiä tietoturvamekanismeja tarvitaan, koska yrityksen tärkeimmät resurssit on turvattava. Tämä tutkielma käsittelee relaatiotietokannanhallintajärjestelmien suojaamista väärinkäytöntunnistamisjärjestelmillä, joissa tunnistaminen perustuu poikkeamiin valtuutettujen käyttäjien tallennetusta normaalikäytöstä.

Nykyisin tietojärjestelmiin on tallennettu suuri osa yritysten ja muiden organisaatioiden tietopääomasta. Sen tuhoaminen, asiaton muuntelu tai joutuminen vääriin käsiin saattavat tuottaa omistajalleen tappioita ja vaikeuksia. Carterin ja Katzin (1996, s. 1) mukaan yrityksen tulevaisuus on sen tietopääomassa, ei niinkään sen käsin kosketeltavassa omaisuudessa. Suurin osa yritysten ja organisaatioiden operatiivisesta tiedosta on tallennettu tietokantajärjestelmiin. Tietokannanhallintajärjestelmistä valtaosa puolestaan perustuu nykyisin relaatiomalliin.

Avoimuuden suosiminen sekä tiedon ja resurssien jakaminen altistavat julkisiin verkkoihin kytketyt tietojärjestelmät rikollisuudelle ja vahingonteolle. Carter ja Katz (1996, s. 2) raportoivat tietojärjestelmiä vastaan tehtyjen hyökkäysten määrän kasvusta. Myös Allen ym. (2000, s. 3) ovat todenneet, että julkisesta Internet-verkosta tulevat organisaatioiden raportoimat hyökkäykset ovat lisääntyneet melkein samassa suhteessa itse verkon yleistymisen kanssa. Harmittomat uteliaat tunnustelijat ovat jokapäiväinen harmi, eivätkä todelliset tunkeutumisyrityksetkään ole enää harvinaisia.

Tietoverkkoihin kytkeytymisestä ja organisaatioiden sekä yksityishenkilöiden välisestä tiedonsiirrosta saatavat hyödyt ovat niin suuret, ettei täydellinen eristäytyminen ole yrityksille enää mahdollista. Arvokkaiden resurssien turvaksi tarvitaankin suojausmenetelmiä, jotka yhtäaikaa sallivat avoimuuden ja toisaalta vartioivat järjestelmien asianmukaista käyttöä. Palomuurit, käyttäjän tunnistus, käyttöoikeusmääritykset ja virustorjunta on otettu käyttöön jo useimmissa tietokoneita käyttävissä organisaatioissa, mutta

näiden lisäksi on tarvetta perinteisiä käyttöoikeusmäärityksiä helpommin hallittavalle ja älykkäämmälle käytönvalvonnalle.

Tunkeutujan- ja väärinkäytöntunnistamisjärjestelmien tutkimusala on vielä nuori, mutta aktiivisen luonteensa ansiosta järjestelmät ovat hyvin lupaavia. Allen ym. (2000, s. 17) ovat kattavasti raportoineet tutkimuksen nykytilaa, joka on keskittynyt pääasiassa verkko-, palvelin- ja käyttöjärjestelmätason tunkeutujantunnistamiseen. Mikään yksittäinen tunnistusjärjestelmä ei kuitenkaan riitä suojaamaan tietojärjestelmää, vaan tarvitaan pienempiä tietojärjestelmän osa-alueita tarkkailevia erikoistuneita järjestelmiä, jotka voidaan sitten fuusioida kokonaisuudeksi. Muiden muassa Forrest ym. (1996, s. 1) puhuvat hajautetun ja erikoistuneen tunkeutujan tunnistamisen puolesta ja vertaavat tietoturvaa immuunijärjestelmään.

Chungin ym. (1998, s. 1) mukaan tunkeutumisen tunnistaminen (intrusion detection) tarkoittaa järjestelmän ulkopuolisten luvattomien sisäänpyrkijöiden tunnistamista, kun taas väärinkäytön tunnistaminen (misuse detection) on laajempi käsite, joka käsittää myös järjestelmän valtuutettujen käyttäjien luvattomat toimet ja käyttöoikeuksien ylitykset. Muiden muassa Lane (1998, s. 24) on todennut, etteivät organisaatioiden ongelmina ole pelkästään ulkopuoliset tunkeutujat, vaan myös organisaation omat jäsenet, jotka saattavat väärinkäyttää järjestelmää; eräiden tutkimusten (Carter ja Katz 1996, s. 3) mukaan yritysten omat työntekijät ovat suurin uhka tietojärjestelmälle.

Tunkeutujan tunnistamisen kaksi päälähestymistapaa ovat Axelssonin (2000, s. 2) mukaan tunnisteperusteinen (signature detection) ja poikkeamaperusteinen (anomaly detection), joista jälkimmäistä tämä tutkielma pääasiassa käsittelee. Tunnisteperusteisessa lähestymistavassa järjestelmään on tallennettu asiantuntijoiden laatimia hyökkäystapojen kuvauksia, joita järjestelmä vertaa suojattavan tietojärjestelmän tapahtumiin. Tunnistaessaan tallennetun hyökkäysmallin järjestelmä ilmoittaa väärinkäytöstä. Tunnisteperusteista tunkeutujan- tai väärinkäytöntunnistamista voisikin verrata virustorjunnan perusajatukseen. Poikkeamaperusteinen järjestelmä pohjautuu puolestaan tunnistamisjärjestelmän opettamiseen ja profiilin rakentamiseen. Järjestelmä oppii opettamisvaiheessa valtuutetun käyttäjän normaalikäytön suojatussa ympäristössä. Tunnistusvai-

heessa järjestelmä havaitsee nykyisessä käytössä ilmenevät poikkeamat profiilista ja ilmaisee tunkeutumisen tai väärinkäytön.

Axelsson (2000, s. 2) toteaa, että poikkeamaperusteisten järjestelmien ongelma on väärien hälytysten suuri määrä. Allen ym. (2000, s. 8) mukaan tällainen tunnistus on väärä positiivinen (false positive), jolloin järjestelmä pitää väärinkäyttönä jotakin täysin vilpitöntä järjestelmän käyttöä. Ongelma johtuu siitä, ettei normaalikäyttöä ole onnistuttu mallintamaan järjestelmään riittävässä laajuudessa. Ongelmana ovat myös tehokkuuskysymykset, sillä laajan normaalikäytön mallin tallettaminen vie paljon tilaa ja malli on tunnistamisvaiheessa hidas käyttää. Mitä suurempaa tieto- ja tapahtumamäärää yritetään kerrallaan käsitellä, sitä käyttökelvottomampi mallista tulee. Toisaalta Axelsson (2000, s. 2) pitää tunnistepohjaisten järjestelmien ongelmana niiden kyvyttömyyttä tunnistaa uudenlaisia tunkeutumisyrityksiä. Allen ym. (2000, s. 8) mukaan tällaiset tunnistamatta jättämiset ovat vääriä negatiivisia (false negatives). Tällainen järjestelmä ei kykene tunnistamaan kuin ne hyökkäykset, joiden kulku järjestelmään on mallinnettu. Toimintatavaltaan uudenlaiset hyökkäykset läpäisevät järjestelmän.

Koska molemmissa päälähestymistavoissa on omat ongelmansa, on löydettävä ratkaisuja, jotka yhdistävät niiden parhaat puolet. Bass (2000, s. 100) on todennut, että millään yksittäisellä tunkeutujantunnistamisjärjestelmällä ei voida taata järjestelmän tietoturvaa, vaan parempi ratkaisu ovat fuusiojärjestelmät, jotka käsittelevät useamman erikoistuneen osatunnistusjärjestelmän tietoa. Myös Hofmeyer ym. (1998, s. 9) mielestä monitasoisista, toisistaan riippumattomista osajärjestelmistä koostuva järjestelmä olisi tehokas ja se ratkaisisi mm. väärien positiivisten aiheuttaman ongelman. Fuusiojärjestelmien toteuttaminen vaatii kuitenkin hyviä osajärjestelmiä ja tämä tutkielma keskittyykin relaatiotietokannanhallintajärjestelmien yhteyteen rakennettaviin väärinkäytöntunnistusjärjestelmiin. Tietokantojen väärinkäytön tunnistamiseen on luontevaa käyttää poikkeamaperusteista lähestymistapaa, koska monimutkaisia rakenteita sisältäville tietokannoille olisi vaikeaa luoda pitäviä tunnisteita, joilla hyökkäysyritykset voitaisiin havaita.

Ensimmäinen tämän tutkielman tutkimusongelma koskee poikkeamaperusteisen tunkeutujan- ja väärinkäytöntunnistamisen tutkimuksessa esitettyjä malleja. Tutkimuksessa

halutaan selvittää, mitä relaatiotietokantojen kontekstissa olennaisia ominaisuuksia näissä yleisissä malleissa on. Myös poikkeamaperusteisen tunnistuksen heikkouksia ja vahvuuksia käsitellään.

Toinen tutkimusongelma koskee sitä, mitkä ovat relaatiotietokannanhallintajärjestelmän yhteyteen rakennettavan väärinkäytöntunnistusjärjestelmän keskeiset ominaisuudet. Tutkimuksella halutaan selvittää, mitä ominaisuuksia tietokantojen yhteyteen rakennettaviin järjestelmiin on aiemmassa tutkimuksessa esitetty, ja mitä muuta yleisessä poikkeamaperusteisessa väärinkäytön tunnistamisen tutkimuksessa on lisäksi pidetty tärkeänä. Erityisesti huomiota kiinnitetään tiedon keruuseen, ominaisuuksien (feature) valintaan ja mallin muodostukseen sekä varsinaiseen käytönaikaiseen tunnistamiseen. Tutkimuksen piiriin eivät kuulu vastatoimenpiteet, harhauttaminen, järjestelmän fyysinen ympäristö, tunnistusjärjestelmän itsensä suojaaminen, tehokkuuskysymykset eivätkä eri relaatiotietokannanhallintajärjestelmien erot.

Kolmanneksi halutaan esimerkin avulla selvittää, miten tällainen järjestelmä voitaisiin käytännössä toteuttaa. Järjestelmästä rakennettavan kevyen prototyypin avulla pyritään osoittamaan tällainen järjestelmä myös käytännössä mahdolliseksi.

Tutkielmalla saavutettiin sille asetetut tavoitteet ja väärinkäytön tunnistamisjärjestelmän rakentamisesta tietokannanhallintajärjestelmän yhteyteen saatiin lisää tietoa. Aiemmasta yleisestä tutkimuksesta kerättiin tietoa tietokannanhallintajärjestelmän yhteyteen sopivista malleista, sekä yhdistettiin niistä saatuja tuloksia tietokannanhallintajärjestelmien kanssa jo kokeiltuihin malleihin. Malleista luotiin yhdistelmä, jossa sekä käyttäjän toiminnan että käyttäytymisen poikkeamien tunnistaminen on selvästi erotettu, mutta kuitenkin toteutettu mallin rakentamisen kannalta tehokkaasti samassa järjestelmässä ja samoilla perustiedoilla. Prototyyppijärjestelmän rakentamisen avulla saatiin paljon tietoa tällaiselle järjestelmälle olennaisesta jäljitystietojen keräämisestä ja sen ongelmista. Prototyypin avulla voitiin myös todeta, että on mahdollista rakentaa järjestelmä, joka käsitteellisesti erottaa ja rakenteellisesti yhdistää toiminnan ja käyttäytymisen poikkeamien tunnistamisen.

Tutkimuksen tuloksia voidaan hyödyntää tutkittaessa poikkeamaperusteista väärinkäytön tunnistamista yleisesti tai tutkittaessa väärinkäytön tunnistamista tietokannanhallintajärjestelmien yhteydessä. Tutkielma tarjoaa perustan myös jatkotutkimukselle ja osoittaa sopivia jatkotutkimuksen kohteita. Tutkielma sopii myös aihepiirin teoreettisiin kysymyksiin ja ratkaisumalleihin tutustuttavaksi lukemiseksi rakennettaessa järjestelmiä tuotantokäyttöön.

Tutkielmassa käsitellään poikkeamaperusteista väärinkäytöntunnistamista esittelemällä ensin alan aiempaa tutkimusta luvussa 2 sekä tarkastelemalla lähestymistavan heikkoja ja vahvoja puolia luvussa 3. Samalla pyritään osoittamaan relaatiotietokantojen yhteydessä oleelliset mallien ominaisuudet. Lisäksi luvussa 4 käsitellään relaatiotietokantojen tunkeutujan- ja väärinkäytöntunnistamisen tutkimusta ja aiemmin esitettyjen mallien ominaisuuksia vertaillaan. Vertailun jälkeen aiempien mallien ominaisuuksia täydennetään tuomalla mukaan hyödyllisiä näkökulmia yleisestä poikkeamaperusteista tutkimuksesta. Luku 5 kuvaa aiemman tutkimuksen pohjalta soveltaen rakennetun prototyyppijärjestelmän koejärjestelyineen, ja luku 6 tarkastelee prototyyppijärjestelmästä saatuja tuloksia.

2 POIKKEAMAPERUSTEISET MENETELMÄT VÄÄRINKÄYTÖN TUNNISTAMISESSA

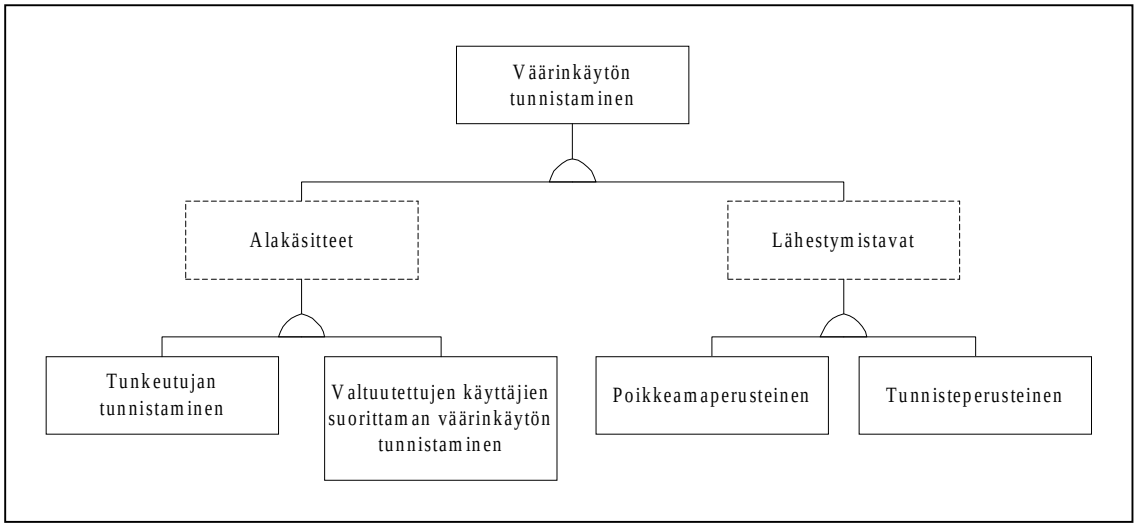
Tunkeutujan- ja väärinkäytöntunnistaminen voi pohjautua tietojärjestelmissä pääperiaatteeltaan joko tunnettujen hyökkäystapojen tunnistamiseen tai normaalikäytöstä poikkeavan toiminnan tunnistamiseen. Poikkeamaperusteinen lähestysmistapa sopii yleisiltä periaatteiltaan luonnollisesti tietokannanhallintajärjestelmän yhteyteen rakennettavan väärinkäytöntunnistamisjärjestelmän lähtökohdaksi. Seuraavassa käsitellään tarkemmin tunkeutujan- ja väärinkäytöntunnistamiseen liittyvää terminologiaa sekä poikkeamaperusteisten tunnistamisjärjestelmien periaatteita.

2.1 Yleiset periaatteet ja käsitteet

Tietojärjestelmien luvatonta käyttöä voidaan tarkastella eri näkökulmista, ja näkökulman valinta vaikuttaa käytettyyn terminologiaan. Suurin osa tunkeutujan tunnistamiseen liittyvästä kirjallisuudesta, esimerkiksi Lee ym. (1999, s. 1) jakaa tunnistamisen poikkeamien tunnistamiseen (anomaly detection) ja väärinkäytön tunnistamiseen (misuse detection). Tällöin väärinkäytöllä tarkoitetaan pelkästään tietojärjestelmän heikkouksien hyväksikäyttämistä ja sen tunnistamisella tunnisteisiin (signature) perustuvaa tunkeutumisen havaitsemista. Axelsson (2000, s. 2) on tunkeutujan tunnistamisen taksonomiassaan erottanut tunkeutujan tunnistamisen päälähestymistavat kahteen luokkaan: poikkeamien tunnistamiseen ja tunnisteiden tunnistamiseen. Näiden lisäksi on käsitelty molempia päälähestymistapoja hyödyntävää mallia eli yhdistelmätunnistamista (compound detection).

Jos järjestelmiin tunkeutumista ja järjestelmien väärinkäyttöä tarkastellaan laajemmasta näkökulmasta, voidaan huomata, että edellisen kaltainen käsitteellinen jako on kuitenkin hieman harhaanjohtava. Chung ym. (1998, s. 1) on määritellyt väärinkäytön käsitteenä huomattavasti paremmin. Chungin ym. (1998, s. 1) mukaan väärinkäytön (misuse) tunnistaminen on tunkeutumisen tunnistamisen ja valtuutettujen käyttäjien suorittaman väärinkäytön tunnistamisen yläkäsite, ja sen eri lähestymistapoja ovat poikkeamaperusteinen (anomaly based) ja tunnisteperusteinen (signature based) väärinkäytön tunnistaminen.

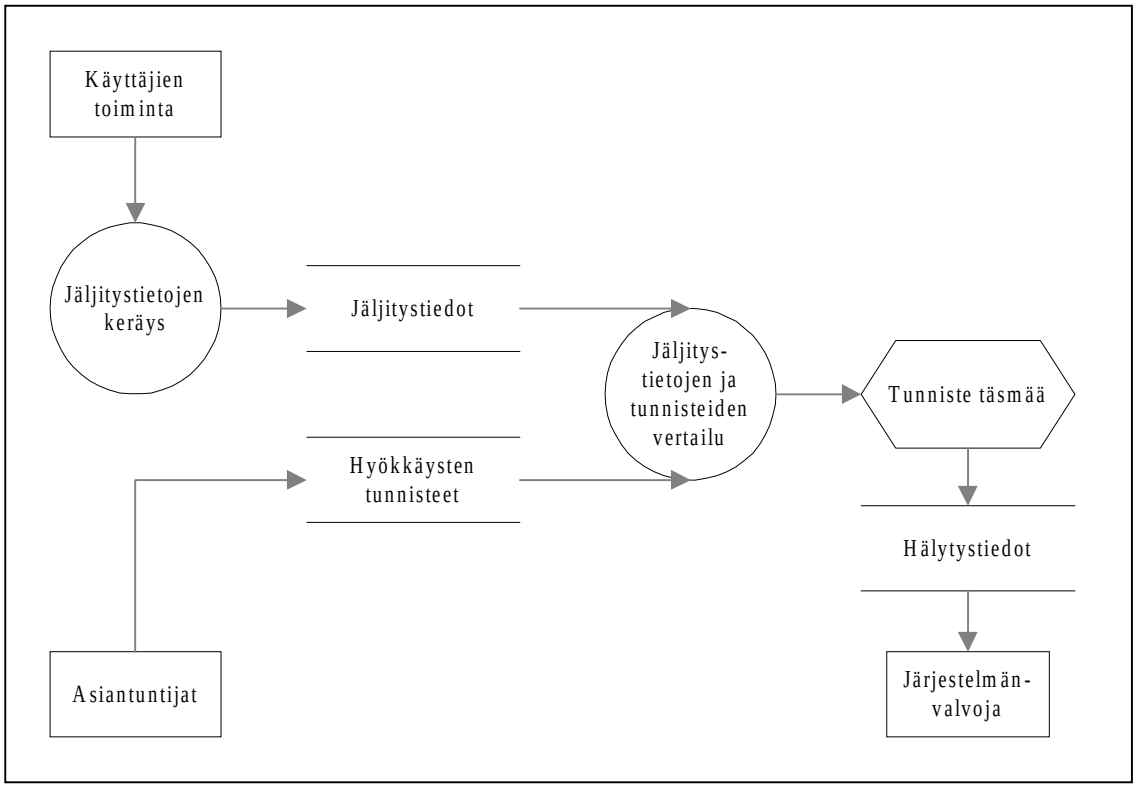
Tämän tutkielman perustaksi on otettu edellä mainittu laajempi näkökulma, jossa tietojärjestelmän väärinkäytöksi luetaan sekä ulkopuolisten tunkeutujien suorittama väärinkäyttö että järjestelmän valtuutettujen käyttäjien suorittama väärinkäyttö. Tämän tutkimuksen omaksuma käsitteellinen jako on havainnollistettu kuviossa (KUVIO 1), joka on piirretty Chungin ym. (1998, s. 1) esittämän jaottelun perusteella. Vaikka näkökulmaero näkyykin lähdemateriaaliin viitattaessa, ei se aiheuta ongelmaa, sillä tämä tutkimus keskittyy poikkeamaperusteiseen väärinkäytön tunnistamiseen, jolloin käsitteiden suhteen ei synny ristiriitaa. Myös pelkästään tunkeutujan tunnistamiseen keskittyvät lähteet ovat käyttökelpoisia koko väärinkäytön tutkimusalueen tarkasteluun, sillä samoja perusajatuksia voidaan soveltaa.



KUVIO 1. Väärinkäytöntunnistamisen käsitteellinen jako. Väärinkäytön tunnistaminen voidaan jakaa tunkeutujien tunnistamiseen ja valtuutettujen käyttäjien suorittaman väärinkäytön tunnistamiseen. Väärinkäytön tunnistamisella on kaksi päälähestymistapaa: poikkeamaperusteinen ja tunnisteperusteinen.

Tunnisteperusteinen väärinkäytön tunnistaminen pohjautuu ennalta mallinnettujen hyökkäysmallien havaitsemiseen tietojärjestelmässä. Axelssonin (2000, s. 6) mukaan tunkeutuja tunnistetaan tällaisessa järjestelmässä tunkeutumisprosessin mallin ja sen tietojärjestelmään jättämien jälkien perusteella. Järjestelmään on ohjelmoitu tunnisteina sen tunnistettavaksi tarkoitettujen väärinkäyttöyritysten mallit ja tällaisen järjestelmän toimintaperiaatetta voisi verrata vaikkapa tietokonevirusten tunnistamiseen tarkoitettuihin ohjelmiin. Asiantuntijoiden ohjelmoimia hyökkäysten tunnisteita vertaillaan järjestelmän tapahtumiin, ja mikäli järjestelmässä havaitaan tunnisteen mukainen hyökkäys, voidaan epäillä väärinkäyttöä. Tunnisteperusteisen järjestelmän tunnistamisen periaate

on esitetty kuviossa (KUVIO 2), joka on piirretty kirjallisuudessa esitettyjen kuvausten pohjalta. Tunnisteperusteinen järjestelmä on oletusarvoisesti kaiken käytön salliva, sillä se tunnistaa väärinkäyttöyrityksiksi vain ne väärinkäytön kuvaukset, jotka siihen on ohjelmoitu.

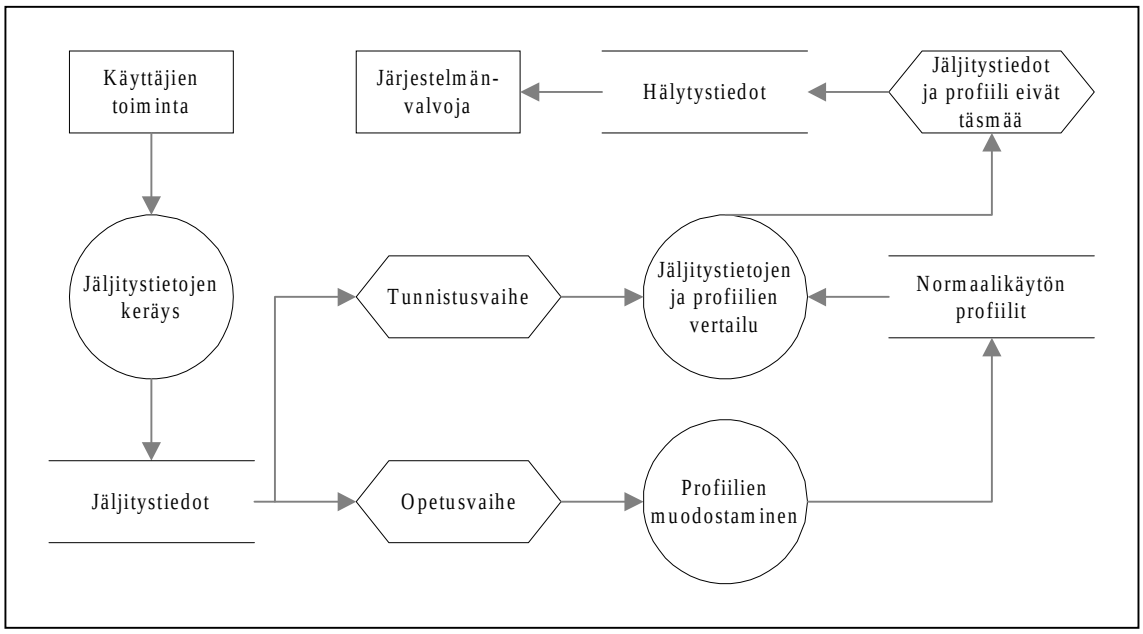


KUVIO 2. Tunnisteperusteisen väärinkäytöntunnistamisen toimintaperiaate. Käyttäjien toiminnasta kerättyjä jäljitystietoja verrataan tietoturva-asiantuntijoiden ohjelmoimiin hyökkäysten tunnisteisiin ja jos ne täsmäävät, annetaan hälytys.

Muiden muassa Lee ja Xiang (2001, s. 1) toteavat, etteivät tällaiset järjestelmät ole tehokkaita uudenlaisia, ennalta tuntemattomia hyökkäyksiä vastaan. Lee ja Xiang (2001, s. 1) perustelevat, että esimerkiksi vuoden 1999 DARPA:n tutkimuksessa vain 70 % testatuista tunkeutumisyrityksistä tunnistettiin, koska suuri osa uusista tunkeutumismalleista jäi järjestelmiltä huomaamatta. Allen ym. (2000, s. 8) määrittelevät tällaisen virheen vääräksi negatiiviseksi. Tunnistepohjaiset järjestelmät ovat enimmäkseen asiantuntijoiden käsin rakentamia ja siksi helposti ajastaan jäljessä.

Poikkeamaperusteinen väärinkäytön tunnistaminen pohjautuu tietojärjestelmän normaalitoiminnan mallintamiseen ennakolta. Poikkeamaperusteisessa järjestelmässä muodostetaan ensin malli normaalikäytöstä ja verrataan sitten tietojärjestelmän tapahtumia

normaalikäytön malliin. Kaikki käyttö, jota ei ole aiemmin tallennettu, on tällaisen järjestelmän kannalta epäilyttävää. (Axelsson 2000, s. 5) Poikkeamaperusteisessa väärinkäytöntunnistamisjärjestelmässä opetetaan ensin järjestelmälle käyttäjän normaalitoimintaa. Tästä normaalitoiminnasta järjestelmä muodostaa profiilin eli käyttäjän normaalitoiminnan mallin. Kun tunnistusjärjestelmä on käytössä, käyttäjän toimia verrataan aiemmin tallennettuun malliin, ja mikäli ne poikkeavat siitä, voidaan epäillä väärinkäyttöä. Kirjallisuudessa esitettyjen mallien perusteella piirretyssä kuviossa (KUVIO 3) on esitetty poikkeamaperusteisen tunnistamisen toimintaperiaate.



KUVIO 3. Poikkeamaperusteisen väärinkäytöntunnistamisen toimintaperiaate. Käyttäjien normaalikäytöstä tallennettuja profiileja verrataan nykykäytöstä kerättyihin jäljitystietoihin, ja jos ne eivät täsmää, annetaan hälytys.

Poikkeamaperusteinen järjestelmä on oletusarvoisesti kaiken käytön kieltävä, sillä se tunnistaa väärinkäyttöyrityksiksi kaiken toiminnan, jota ei ole mallinnettu normaaliksi. Lee ja Xiang (2001, s. 1) arvostelevat olemassa olevia poikkeamaperusteisia järjestelmiä siitä, että ne usein virheellisesti tunnistavat täysin vilpittömän käytön väärinkäytöksi. Tällaiset väärät positiiviset ovat Allen ym. (2000, s. 8) mukaan tilanteita, joissa järjestelmä tunnistaa tunkeutumisen tapahtuneen, vaikka todellisuudessa näin ei ole.

Väärinkäytöntunnistamista varten tarvitaan kielioppi, jonka varaan tiedon keruu ja tunnistaminen voidaan rakentaa. Denning (1987, s. 2) on määritellyt yleisen poikkeamaperusteisen järjestelmän kuusi komponenttia:

- Tekijät (subjects) ovat toiminnan (action) käynnistäjät tietojärjestelmässä – esimerkiksi käyttäjä tai toinen prosessi. Kaikki toiminta tapahtuu tekijöiden toimesta.
- Kohteet (objects) ovat järjestelmän resursseja – esimerkiksi tiedosto, komento tai laite. Järjestelmän sovellusalueesta riippuen mallinnettu tarkkuus voi olla erilainen, sillä esimerkiksi tietokantasovellus voi vaatia tietuetasoista seurantaa, kun taas useimmille sovelluksille riittää tiedostotaso.
- Jäljitystietueet (audit records) luodaan, kun tekijä käynnistää jonkin toiminnan kohteeseen liittyen.
- Profiilit (profiles) kuvaavat tekijöiden normaalia käyttäytymistä kohteiden suhteen, joidenkin tilastollisten metriikoiden ja havaitun toiminnan perusteella.
- Poikkeamatietueet (anomaly records) luodaan, kun poikkeava käyttäytyminen havaitaan.
- Toimintasäännöt (activity rules) kuvaavat, mitä tehdään, kun jokin ehto täyttyy.

Denning (1987, s. 2) ei ole tarkemmin määritellyt toimintaa erilliseksi komponentiksi, mutta toteaa, että toimintaa ovat esimerkiksi kirjautumiset, komentojen ja ohjelmien suoritukset sekä tiedostojen ja laitteiden haut. Kaksi ensimmäistä Denningin (1987, s. 2) esittämistä kuudesta komponentista, eli tekijä sekä toiminnan kohde, ovat selvästi jäljitystiedoista kerättäviä ja mallin rakentamiseen tarkoitettuja osia. Tekijä, toiminta ja toiminnan kohde täytyy mallintaa, jotta tiedetään kuka tekee, mitä tekee ja mihin tekeminen kohdistuu. Tällainen malli on tuttu esimerkiksi tietokannan käyttöoikeuksien määrittelystä (Bertino ym. 1999, s. 105). Tietysti malli voi olla yksinkertaisempikin, jolloin jätetään esimerkiksi tekijä tai toiminnan kohde pois. Lisäksi tunnistamisjärjestelmään voidaan tallentaa tietoa esimerkiksi ajasta, järjestelmän palauttamista poikkeuksista ja resurssikäytöstä, kuten Denning (1987, s. 3) ehdottaa. Tällainen resurssikäyttöön perustuva tunkeutujan tunnistamisjärjestelmä on esimerkiksi NIDES, joka Andersonin ym. (1995, s. 3) mukaan käyttää profiilin muodostamiseen myös esimerkiksi järjestelmän suorittimen käyttöaikaa. Seleznyov ja Puuronen (1998, s. 7) ovat ehdottaneet käy-

tettäväksi myös paikan ja ajan suhteita. Denningin kieliopin (1987, s. 2) neljä muuta komponenttia ovat puolestaan tunnistamisjärjestelmän rakenteellisia osia.

Poikkeamaperusteisen tunnistuksen pohjana olevan mallin kannalta on erittäin tärkeää määritellä tarkoin, mitä halutaan mallintaa, sillä Lanen (1998, s. 10) mukaan mallin tarkkuus on suhteessa sen käyttämiin resursseihin. Samalla Lane (1998, s. 10) toteaa, ettei tunnistamisjärjestelmä saa häiritä muuta järjestelmää ajan tai muun resurssikäytön suhteen.

Ilgun ym. (1995, s. 12) keräävät sääntöperusteisessa (rule-based) STAT-järjestelmässään tekijöiden, toiminnan ja kohteiden välisiä suhteita ja mallintavat ne sääntöinä. Lisäksi esimerkiksi Chung ym. (1999) ovat tutkimuksessaan keskittyneet tekijöiden, kohteiden ja toiminnan mallintamiseen eli normaalikäytön malli muodostetaan sen mukaan, kuka järjestelmällä tekee, mitä järjestelmällä tehdään ja mihin tekeminen kohdistuu. Toisaalta tekemisen kohde mallinnetaan tällaisessa mallissa usein hyvin hienojakoiseksi. Edellä mainitun kaltaiset järjestelmät pystyvät tunnistamaan poikkeamat normaalikäytöstä sen perusteella, mitä järjestelmällä tehdään, eli mallinnettavan kokonaisuuden muodostavat tekijän kohteille suorittamat toiminnot. Tällöin poikkeamaperusteinen väärinkäytön tunnistamisjärjestelmä pyrkii tunnistamaan, tekeekö käyttäjä järjestelmän kohteille normaaleiksi mallinnettuja toimia.

Toinen lähestymistapa on mallintaa tekijän käyttäytymistä laajemmin ja pyrkiä tunnistamaan tekijä käyttäytymisen perusteella. Tällöin voidaan käyttää esimerkiksi tilastollisia määreitä. Denning (1987, s. 3) on ehdottanut käytettäväksi tilastolliseksi suureeksi mm. suorittimen käyttöastetta, tulostettujen sivujen määrää tai istunnon kestoaikaa. Samanlainen käyttäjän käyttäytymistä mallintava suure voisi olla myös esimerkiksi Luntin (1988, s. 6) mainitsema näppäilynopeus. Tällaisia lähestymistapoja hyödyntävät mallit pyrkivät tallennettua normaalikäyttäytymistä ja havaittua uutta käyttäytymistä vertailemalla päättelemään, onko käyttäjä todella se, joka väittää olevansa.

Forrest ym. (1996) ja Lane (1998) ovat pyrkineet löytämään käyttäjän käyttäytymismallin suoraan järjestelmään syötettyjen komentojen sarjasta. Jäljitystiedon rakennetta ei ole semanttisessa mielessä käytetty hyväksi, vaan komentojen ja valitsimien ajallinen

(temporal) järjestys tallennettavissa sarjoissa määrää käyttäjän käyttäytymisen. Mallit ovat hyvin pelkistettyjä ja esimerkiksi Lanen (1998) mallissa tiedostonimet eli kohteet on korvattu vain määrillä, ja malli koskee kerrallaan vain yhtä käyttäjää.

Jotta väärinkäytön tunnistamiseen tarkoitettujen mallien eroja voidaan arvioida, täytyy ensin tarkastella, mitä halutaan mallintaa ja myöhemmin tunnistaa. Käyttäjän varsinaiset toimet kohteiden suhteen tulee erottaa käyttäjän käyttäytymismallien tunnistamisesta. Käyttäjän toimien mallintaminen on lähellä käyttöoikeuksien hallintajärjestelmien perusajatusta, kun taas käyttäjän käyttäytymisen mallintaminen perustuu biometriaan. Käyttäytymisen mallintaminen johtaa luonnollisesti huomattavasti monimutkaisempiin ja laskennallisesti raskaampiin malleihin kuin pelkkä käyttäjän toiminnan mallintaminen.

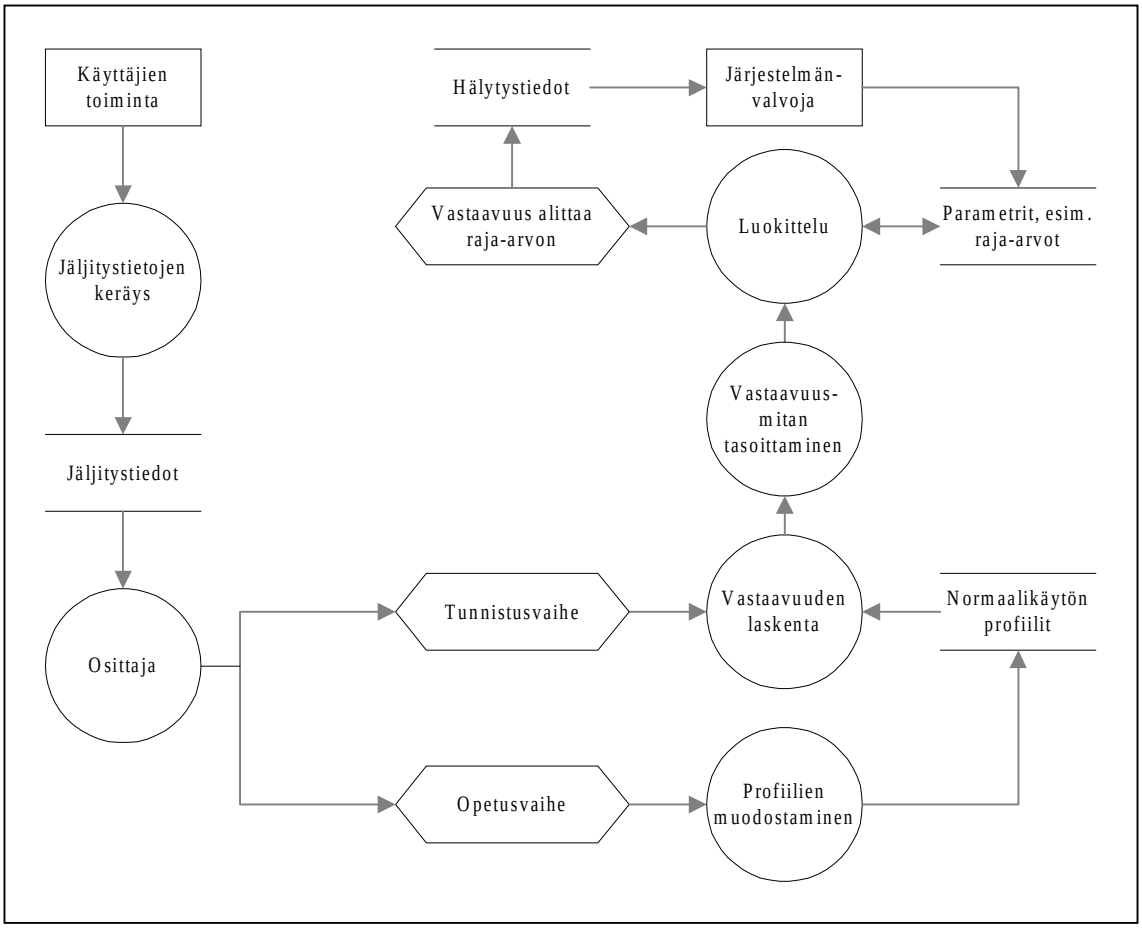
Edellä on esitetty poikkeamaperusteisen tunnistamisen perusperiaatteet, joita syvennetään myöhemmin. Seuraavassa luvussa käsitellään tarkemmin poikkeamaperusteisen väärinkäytöntunnistusjärjestelmän yleistä rakennetta.

2.2 Poikkeamaperusteisen väärinkäytöntunnistusjärjestelmän rakenne

Poikkeamaperusteisen tunkeutujan- ja väärinkäytöntunnistamisjärjestelmän fyysistä rakennetta on vielä syytä tarkastella lähemmin kokonaiskuvan muodostamiseksi. Esimerkkinä käytetään soveltaen Lane ja Brodley (1999, s. 300) mallia (KUVIO 4), jossa voidaan erottaa tietoa keräävät, profiileja muodostavat sekä profiileja ja nykykäyttöä vertailevat osat.

Väärinkäytöntunnistamisjärjestelmän rakentaminen alkaa käyttäjän normaalitoiminnan mallintamisella. Käyttäjän toiminnasta kerätään valvotuissa oloissa riittävästi näytteitä sopivaa tietolähdettä, kuten järjestelmän jäljitystietoja käyttäen. Aluksi järjestelmään täytyy määritellä, miten ja mistä jäljitystietoja kerätään. Raaka jäljitystieto jalostetaan jollakin tavalla sopivaan muotoon ennen profilointia, jotta siitä saadaan kaikki tarpeelliset ominaisuudet (feature) tehokkaasti talteen. Lanen ja Brodleyn (1999, s. 300) mallissa UNIX-komentosarjoista koostuvaa jäljitystietoa jalostetaan osittamalla se lyhyisiin jaksoihin, poistamalla siitä tiedostonimet ja korvaamalla ne ainoastaan tiedostojen mää-

rää kuvaavalla tunnuksella. Usein jäljitystietoja joudutaan jalostamaan myös muilla tavoin. Jokin osa järjestelmästä voi huolehtia muun muassa puuttuvista arvoista ja laskea tarvittavat tilastolliset suureet.



KUVIO 4. Poikkeamaperusteisen tunkeutujantunnistamisjärjestelmän rakenne Lane ja Brodley (1999) mukaan. Jäljitetty käyttäjän toiminta opetus- ja tunnistusvaiheessa muutetaan osittajassa vakiomittaisiksi jaksoiksi. Opetusvaiheessa profiilin muodostaja tallentaa jaksoista profiilit ja tunnistusvaiheessa ositettuja tietoja verrataan profiileihin. Lasketut yksittäiset vastaavuudet tasoitetaan ja vastaavuuksien joukosta päätellään, onko toiminta pidemmällä aikavälillä epäilyttävää.

Profiilin muodostaja lukee esikäsitellyn tiedon käyttäytyjän normaalikäyttäytymisestä ja muodostaa siitä jonkin mallin mukaisesti käyttäjäkohtaiset profiilit, jotka se tallettaa tietokantaan. Profiilin muodostaja voi olla yksinkertainen tietoa siirtävä ja hieman erilaiseen muotoon tallentava ohjelmanosa tai hyvin monimutkainen, esimerkiksi puurakenteita tai verkkomalleja rakentava osajärjestelmä. (Lane ja Brodley 1999, s. 300)

Varsinaisessa järjestelmän käyttövaiheessa käyttäjän toiminnasta muodostetaan esikäsiteltyä tietoa aivan kuten opetusvaiheessakin. Tämä tieto syötetään sitten vastaavuuden laskentaan, joka jotain tapaa käyttäen laskee, missä määrin havaitut uudet ja tietokannassa olevat historialliset tiedot muistuttavat toisiaan. Vertailu voi olla yksinkertaisesti profiilin tietoja ja uusia jäljitystietoja vertailevaa tai esimerkiksi tilastolliseen laskentaan perustuvaa. Lanen ja Brodleyn (1999, s. 300) mallissa vertailtavat profiilit ja uudet jäljitystiedot ovat komentojen ajallisesti järjestettyjä ja lomitetusti ositettuja sarjoja. Mikäli uusi jäljitystieto ei järjestelmän tunnistusperiaatteen mukaan täsmää profiilissa olevien tietojen kanssa, toiminta voidaan merkitä epäilyttäväksi. (Lane ja Brodley 1999, s. 300)

Koska väärät hälytykset halutaan minimoida, yksittäisiä eroavaisuuksia ei usein raportoida hälytyksinä, vaan esimerkiksi tietyn mittaisen jaksojen sarjan vastaavuuksien keskiarvo lasketaan vastaavuusmitan tasoittamiseksi. Tätä tasoitettua vastaavuusmittaa verrataan sitten asiantuntijoiden asettamiin raja-arvoihin, jotta voitaisiin päättää, onko havaittu sarja niin erilainen, että hälytys on aiheellinen. Mikäli raja-arvo ylittyy, hälytys annetaan asiantuntijoiden tarkastettavaksi. Pieniä eroavaisuuksia voidaan käyttää palautesilmukan (feedback loop) avulla profiilin jatkuvaan opettamiseen tai järjestelmäparametrien säätöön. (Lane ja Brodley 1999, s. 300)

Edellä esitetty yksinkertaistettu malli on tarkoitettu kokonaiskuvan saamiseen poikkeamaperusteisen väärinkäytöntunnistamisjärjestelmän toiminnasta. Kaikkia yksityiskohtia ei ole kuvattu, eikä poikkeamien tunnistaminen tai järjestelmän uudelleen opettaminen ole aivan yhtä suoraviivaista kuin edellä on kuvattu. Periaatteellisten ja rakenteellisten erojen lisäksi jäljitystiedon keräämiseen, sen esiprosessointiin, normaalikäytön mallin muodostamiseen ja poikkeamien tunnistamiseen on poikkeamaperusteisessa väärinkäytön- ja tunkeutujantunnistamisjärjestelmien tutkimuksessa esitetty useita erilaisia tapoja, joita tarkastellaan seuraavassa kohdassa.

2.3 Käyttäjän toimien jäljittäminen ja jäljitystiedon kerääminen

Pystyäkseen erottamaan väärinkäytön tietojärjestelmän normaalikäytöstä, tunnistamisjärjestelmä tarvitsee tietoa käyttäjän toimista. Normaalikäytön profiilien automaattista rakentamista varten tarvitaan tietoa valtuutetun käyttäjän tavallisesta käytöstä. Käyttäjän

toimista pitää saada vastaavaa tietoa myös tunnistamisjärjestelmän ollessa käytössä, jotta nykyistä käyttöä voidaan verrata normaalikäytöstä rakennettua profiilia vastaan. Tietoa hankitaan yleensä järjestelmän lokitiedoista, erillisistä mittaavista apuohjelmista ja erityisistä järjestelmän jäljitystietoa sisältävistä tietokannoista (audit trail).

Chungin ym. (1998, s. 1) mukaan järjestelmän tapahtumia ja käyttäjän toimia voidaan tarkastella ja jäljittää verkko-, palvelin- tai sovellustasolla, joista vain sovellustaso on riittävän monipuolinen, jos halutaan tunnistaa tietoa vastaan tehtyjä hyökkäyksiä. Verkkopohjainen (network based) tunkeutujantunnistamisjärjestelmä voi käyttää tunnistamisen perustana esimerkiksi TCP-liikenteestä muodostettu lokia (Frank 1994, s. 6). Esimerkiksi Lee ym. (1999, s. 9) ovat keränneet verkosta jäljitystietoja ja tallentaneet tiedon mm. tapahtuman ajasta, kestosta, käytetystä palvelusta, lähde- ja kohdepalvelimesta sekä lähetetyistä ja vastaanotetuista tavuista. On selvää, ettei näin matalalla tasolla jäljitystietoja keräävä järjestelmä pysty suorittamaan myöskään tunnistamista kovin monipuolisesti.

Palvelinpohjainen (host based) tunkeutujan- tai väärinkäytöntunnistamisjärjestelmä, kuten esimerkiksi Lanen (1998, s. 28) järjestelmä, käyttää hyväkseen palvelimen käyttöjärjestelmän muodostamia lokeja ja muodostaa profiilit ohjelmien käytöstä niiden perusteella. Käytännössä Lanen (1998) järjestelmä kerää kaapattuja käyttäjän syöttämiä komentojen sarjoja. Verkkoperusteiseen järjestelmään verrattuna käyttäjän antamien komentojen analysointi sisältää enemmän semantiikkaa, mutta tällainenkaan järjestelmä ei pysty kovin hienojakoisesti erottamaan, mitä järjestelmässä tapahtuu. Tunnistamisjärjestelmä voi myös yhdistellä tietoa useista lähteistä. Axelssonin (2000, s. 10) mukaan tiedon kerääminen voidaan lisäksi luokitella vielä joko yhteen tapaan ja paikkaan keskitettyyn (centralised) tai hajautettuun (distributed) ja useilta sensoreilta tietoa yhdistävään tapaan. Axelsson (2000, s. 12) on koonnut useiden olemassa olevien tunkeutujantunnistamisjärjestelmien ominaisuuksia, ja suurin osa mainituista järjestelmistä käyttää palvelinpohjaista lähestymistapaa.

Allen ym. (2000, s. 61) ovat todenneet, että verkkopohjaiset järjestelmät ovat kyvyttömiä analysoimaan kaikkea niiden käsiteltäväksi tarkoitettua liikennettä, koska matalan tason mallissa tallennettavaa tietoa on paljon ja koko ajan kasvava verkkoliikenteen

nopeus vaikeuttaa tietojen keräämistä. Allenin ym. (2000, s. 61) mielestä raakadataa analysoivat verkkopohjaiset järjestelmät eivät myöskään pysty saamaan kokonaiskuvaa siitä, mitä palvelimilla itse asiassa tapahtuu. Monet muut tutkimukset tukevat tätä näkemystä ja ainakin poikkeamaperusteisen järjestelmän kannalta verkko on ympäristönä liian monimutkainen ja käsitteellisesti matalalla tasolla. Chung ym. (1998, s. 2) ovat sitä mieltä, että käyttöjärjestelmän tasolla toimivien palvelinpohjaisten väärinkäytöntunnistamisjärjestelmien lisäksi tarvitaan sovelluskohtaisia tunnistamisjärjestelmiä, jotka pystyvät käsittelemään myös tiedon ja tapahtumien semanttisia suhteita. Toisaalta tulee suhtautua myös hieman kriittisesti tapahtumien jäljitykseen tietotasolla, jolloin tallennettavaa muodostuu yhtä lailla liikaa.

Lunt (1988) on tutkimuksessaan kartoittanut joitakin perinteisimpiä lähestymistapoja tiedon keräämiseksi käyttäjän toimista ja järjestelmästä. Järjestelmän suorittimen, levyjen ja muistin käytön mittaaminen, kirjautumisajat, istuntojen kestot sekä käyttäjän henkilökohtaisiin ominaisuuksiin pohjautuva käyttäjän identifiointi ovat tutkimuksen alkuvaiheissa olleet useiden järjestelmien osana. Tällainen tieto ei kuitenkaan sisällä mitään varsinaista merkitystä, vaan ainoastaan laskee tilastollisia todennäköisyyksiä valituille suureille. Järjestelmä, joka tallettaa kaiken tällaisen tiedon tilastollista analyysiä varten, on välttämättä rakenteeltaan tarpeettoman monimutkainen ja suorituskyvyltään käytännössä kelvoton. Allen ym. (2000, s. 73) arvostelevat esimerkiksi Luntin (1988, s. 15) aikoinaan lupaavaksi mainitsemaa IDES-järjestelmää normaalikäytön mallin rakentamisen vaikeudesta sekä korkeasta väärien positiivisten tunnistusten määrästä, kun käyttäjän toiminta muuttuu.

Denning (1987, s. 3) on esittänyt, että yleisen poikkeamaperusteisen tunkeutujantunnistamisen mallin perustana käytettävässä jäljitystietueessa olisi kuusi attribuuttia: tekijä, toiminta, kohde, järjestelmän palauttama poikkeus, resurssikäyttöluettelo ja aikaleima. Näistä resurssikäyttöluettelo sisältäisi esimerkiksi palautettujen tai kirjoitettujen rivien määrän tai suorittimen käyttöasteen. Keskeisiä elementtejä ovat kuitenkin tekijä, toiminta ja sen kohde. Monimutkaiset toiminnot esiprosessoidaan mallissa siten, että kussakin jäljitystietueessa on vain yksi kohde, mikä yksinkertaistaa käsittelyä. Denning pitää myös tärkeänä, että järjestelmän opettamisessa ja tunnistamisessa käytetään stan-

dardimuotoista tietuetta, koska erimuotoiset jäljitystietueet aiheuttavat ongelmia järjestelmän siirrettävyydelle. (Denning 1987, s. 5)

Lupaavana tunkeutujantunnistamisjärjestelmän lähestymistapana Allen ym. (2000, s. 74) pitävät poikkeamaperusteisia malleja, joissa jäljitystiedosta kerätään järjestelmäkutsujen sarjoja, jotka muodostavat tietojärjestelmän normaalitoiminnan mallin. Tällaisia malleja ovat esittäneet esimerkiksi Forrest ym. (1996), Lane (1998) ja Hofmeyer ym. (1998). Lanen (1998, s. 28) tunnistamisjärjestelmässä käyttäjän antamat komennot kerätään UNIX-käyttöjärjestelmästä käyttämällä erityistä jäsennintä (parser), joka muuntaa järjestelmän tuottaman raakadatan varastointia ja vertailua varten sopivaan muotoon. Esimerkiksi komennot, valitsimet ja niiden järjestys säilytetään, mutta tiedostonimet korvataan kohdetiedostojen lukumäärällä. Jäsennin merkitsee myös kunkin istunnon alun ja lopun. Jäsennetyt komennot ositetaan vakiomittaisiksi ja päällekkäisiksi ja kerätään käyttäjittäin profiileiksi. Forrestin ym. (1996) ja Hofmeyerin ym. (1998) mallit ovat periaatteeltaan samanlaisia, mutta ne keräävät tietoa järjestelmäprosesseista (privileged processes) ja luovat jokaiselle prosessille mallin siitä, millaisia järjestelmäkutsuja se tavallisesti esittää. Käyttämällä järjestelmäkutsuja jäljittämisen perustana voidaan Hofmeyerin ym. (1998, s. 2) mukaan vähentää käyttäjän käyttäytymisen vaihtelun aiheuttamaa epävarmuutta, koska prosessien esittämät järjestelmäkutsut vaihtelevat vähemmän kuin yksittäisen käyttäjän toiminta.

Käyttäjän toimien jäljittämisessä on pyrittävä selkeään ja yhdenmukaiseen esityksen rakenteeseen, jolla pystytään kuvaamaan kokonaisuudessaan järjestelmän kannalta oleellinen käyttö opetus- ja koetusvaiheessa sekä tunnistamisen aikana. Yhdenmukaiseen esitykseen päästään tunnistamalla poikkeamaperusteisen järjestelmän kannalta tärkeimmät jäljitettävät ominaisuudet ja täydentämällä niitä aina kunkin sovellusalueen mukaan olennaisilla lisätiedoilla.

Denningin (1987, s. 3) esittämän yleisen poikkeamaperusteisen tunkeutujan tunnistamisen mallin mukaan yksi keskeinen jäljitystiedon attribuutti on tekijä eli tieto toiminnan käynnistäneestä käyttäjästä tai järjestelmän prosessista. Tieto tekijästä tarvitaan, jos halutaan muodostaa käyttäjäkohtaisia profiileja. Käyttöoikeusjärjestelmien tapaan myös väärinkäytöntunnistuksessa pitäisi olla mahdollista sallia eri käyttäjille erilaisia

profiileja, sillä myös todellisessa maailmassa käyttäjien työtehtävät ja oikeudet tiedon saannin ja muokkaamisen suhteen vaihtelevat. Tähän liittyen toinen keskeinen tallennettava jäljitystiedon ominaisuus on toiminta eli käyttäjän suorittama operaatio. Väärinkäytön tunnistamisen kannalta on merkittävää, millaista toimintoa käyttäjä yrittää suorittaa. Ainakin luonti-, kirjoitus-, luku-, tuhoamis- ja ajotoiminnot ovat luonnollisesti tärkeitä seurattavia. Kolmas oleellinen attribuutti on toiminnan kohde eli järjestelmän resurssi, joka voi olla esimerkiksi tiedosto, hakemisto, komento tai tietokannan taulu. Esimerkiksi Ilgunin ym. (1995, s. 12) malli jäljitystiedon keräämisestä tukee Denningin näkemystä. Myös tässä mallissa jäljitystietueen perusrakenteen muodostavat tekijä, toiminta ja kohde.

Ajan tallentaminen on Denningin (1987) mallissa yksi jäljitystiedon perusosa. Ajan tallentamisen tai ajallisen järjestyksen määräävän tiedon keräämisen hyödyllisyys riippuu siitä, halutaanko järjestelmän mallintavan vain käyttäjän tavallista toimintaa vai myös käyttäjän tavallista käyttäytymistä. Mikäli normaalitoiminnan mallia rakennetaan kuten Chungin ym. (1999) järjestelmässä, ei tapahtumien ajallisella järjestyksellä ole juurikaan merkitystä. Toisaalta taas Lanen (1998) mallissa tapahtumien järjestys on mallin ja tunnistamisen perusta, koska sen perusteella halutaan tunnistaa käyttäjän normaalia käyttäytymistä biometrian tapaan. Käyttäytymisen mallintamiseen tarvitaan Lanen (1998) järjestelmässä tapahtumien ajallinen järjestys, jolloin tapahtumista voidaan muodostaa ajallisesti järjestettyjä käyttäytymistä kuvaavia jaksoja. Seleznyov ym. (1999) hyödyntävät tapahtumien välisiä ajallisia suhteita vieläkin tarkemmin. Mikäli aika sellaisenaan sisällytetään osaksi mallia, on se esitettävä sopivasti diskreetiksi muunnettuna. Tapahtumien ajalla on aina myös yleistä informatiivista merkitystä, vaikka sitä ei varsinaisesti mallin muodostamisessa käytettäisikään.

Määrällisten tietojen kerääminen riippuu sovellusalueesta. Kuten edellä todettiin, tilastollisten elementtien sisällyttäminen tekee kuitenkin mallista monimutkaisen, eikä todella merkittäviä ja ristiriidattomia suureita ole helppo valita.

Tärkeitä Denningin (1987) mallissa mainittuja jäljitystiedon keräämisen kohteita ovat järjestelmän palauttamat poikkeukset. Järjestelmässä on usein jo olemassa käyttöoikeuksia valvovia järjestelmiä, jotka palauttavat virheilmoituksia, kun käyttäjä suorittaa

niiden kannalta laittomia toimintoja. Tällaisten ilmoitusten havaitseminen väärinkäytöntunnistusjärjestelmässä tehostaa järjestelmää, kun jo valmiiksi olemassa olevia tietoturvamekanismeja voidaan integroida väärinkäytön tunnistamiseen. Esimerkiksi käyttöjärjestelmien käyttöoikeusmääritykset ja tietokannanhallintajärjestelmien valtuutusjärjestelmät (authorization systems) voisivat toimia poikkeustiedon lähteinä.

Denning (1987, s. 5) mainitsee jäljitystiedon muodostamisen ongelmaksi sen sopivan tason määrittämisen. Liian yksityiskohtaisten jäljitystietojen muodostaminen häiritsee varsinaista järjestelmää, ja tiedon suuri määrä saattaa ylikuormittaa myös tunnistamisjärjestelmän. Chung ym. (1999, s. 5) ovat sitä mieltä, että kaiken jäljittämistä ei pitäisi tavoitella, vaan tietoturvasta vastaavan henkilön tulisi voida valita järjestelmän tärkeimmät jäljitettävät ominaisuudet.

Jäljitystiedon keräämisen kohteet ja laajuus määräävät osittain käyttäjän toimista rakennettavan normaalikäytön mallin ilmaisuvoiman ja tehokkuuden. Tämän vuoksi tunnistamisjärjestelmässä pitäisi olla hyvät mahdollisuudet jäljitystiedon keräämisen määrittelemiseen. Jäljitystietojen keräämisen monipuoliset mahdollisuudet eivät tosin tarkoita, että kaikkea toimintaa on jäljitettävä. Ne antavat vain mahdollisuuden käyttää erilaisia rakenneratkaisuja ja jäljitystiedon ominaisuuksia mallin muodostukseen. Tärkeimmät tallennettavat jäljitystiedon ominaisuudet poikkeamaperusteisen mallin luomiseksi ovat edellä käsitellyn perusteella tekijä, toiminta ja toiminnan kohde. Seuraavassa kohdassa on tarkasteltu, kuinka jäljitystiedoista voidaan rakentaa normaalitoiminnan tai –käyttäytymisen malleja.

2.4 Normaalitoiminnan ja -käyttäytymisen malli ja profiilien muodostaminen

Käyttäjän jäljitystiedoista muodostetaan tunnistusjärjestelmän opetusvaiheessa joko käyttäjä-, käyttäjäryhmä- tai roolikohtaiset profiilit jonkin mallin mukaisesti. Joissakin malleissa profiili voi olla myös ohjelma- tai järjestelmäkohtainen. Mallin käyttötapa eli väärinkäytöntunnistamisen periaate vaikuttaa mallin muotoon. Axelsson (2000, s. 5) jakaa taksonomiassaan oppivat järjestelmät aluksi kahteen pääryhmään: aikajaksolliseen (time series) ja aikajaksottomaan (non-time series). Aikajaksottomiin malleihin kuuluvat sääntöjä mallintavat (rule modelling) ja kuvailevat tilastolliset (descriptive

statistics) järjestelmät. Mallin muodostamisessa aikaa hyödyntävistä lähestymistavoista Axelsson mainitsee esimerkiksi neuroverkot ja ”muut enemmän tai vähemmän eksoottiset mallinnustekniikat”. Esimerkiksi Forrestin ym. (1996) ja Hofmeyerin ym. (1998) esittämät järjestelmäkutsuista malleja luovat järjestelmät hyödyntävät aikaa mallin luomisessa, sillä mallin perusta on komentojen ajallinen järjestys.

Mikäli jäljitystietojen perusteella muodostetaan sääntöpohjainen profiili, se voi olla kohtuullisen yksinkertainen ja pieni. Jos malliin lisätään tilastollisia suureita ja poikkeamat lasketaan niiden perusteella, malli muuttuu monimutkaisemmaksi, sillä tilastollisen laskennan suorittamiseksi täytyy kerätä historiatietoa, eikä mallia voi samalla tavoin tiivistää säännöiksi. Myös ajan mallintaminen kasvattaa mallia. Lane (1998, s. 22) toteaakin, että tällaisille malleille täytyy niiden käytännöllisyyden vuoksi asettaa joitakin rajoituksia.

Denningin (1987, s. 5) mukaan profiili kuvaa tekijän käyttäytymistä eri kohteiden suhteen ja tätä käyttäytymistä mitataan erilaisilla tilastollisilla metriikoilla. Denningin (1987, s. 7) mallissa profiili on rakenteeltaan kymmenkohtainen tietue. Profiilitietueita luodaan opetusvaiheessa jäljitystietueiden perusteella. Mallin mukaan profiiliin tallennetaan tietoa seuraavasti:

- Muuttujanimi (variable-name) identifioi profiilitietueen yhdessä tekijän ja kohteen kanssa.
- Toimintokaava (action-pattern) kuvaa tietueeseen liittyvät toiminnot, esim. luku ja ajo.
- Poikkeuskaava (exception-pattern) kuvaa tietueeseen liittyvät poikkeukset.
- Resurssikäyttökaava (resource-usage-pattern) kuvaa mitatun resurssikäytön.
- Aikajakso (period) määrää mittauksen kestoajan.
- Muuttujatyyppi (variable-type) määrää miten tietoa laskennassa käsitellään.
- Raja-arvo (threshold) on raja, joka edellisen tyypin mukaan määrää poikkeaman.
- Tekijäkaava (subject-pattern) kuvaa tietueeseen liittyvät tekijät.
- Kohdekaava (object-pattern) kuvaa tietueeseen liittyvät kohteet.
- Arvo (value) on viimeisimmän havainnon muoto ja arvo.

Kaavatyyppiset attribuutit voivat olla moniarvoisia, eli yksi profiilitietue voi sisältää esimerkiksi monta tekijää, kohdetta tai toimintoa. Denning (1987, s. 6) ehdottaa tilastollisesti kuvailevien tietojen käsittelyyn viittä eri tapaa:

- Operationaalisessa mallissa (operational model) on empiirisesti valittu raja-arvo, jonka ylittyessä toiminta on epänormaalia.
- Keskiarvo- ja keskihajontamallissa (mean and standard deviation model) toiminta on epänormaalia, jos se ei mahdu luottamusvälille.
- Monimuuttuja-analyysimallissa (multivariate model) vertaillaan kahden tai useamman muuttujan välisiä riippuvuuksia keskipoikkeamamallin pohjalta.
- Markovin prosessimallissa (Markov process model) vertaillaan järjestelmän tilan muutoksia matriiseina.
- Aikasarjamallissa (time series model) käsitellään myös järjestelmän tapahtumien välisiä aikasuhteita ja tapahtumien todennäköisyydet perustuvat niiden ajalliseen järjestykseen.

Tilastollisten suureiden käyttö on Denningin mallissa pääosassa ja niillä mallinnetaan käyttäjän käyttäytymistä. Tämä lähestymistapa on innostanut tunnetun ja paljon tutkitun NIDES-järjestelmän rakentamiseen.

NIDES-järjestelmä tunnistaa järjestelmän ohjelmien epätavallista käyttäytymistä käyttämällä mittareinaan muun muassa käyttäjän varaamaa suoritinaikaa, järjestelmän käyttämää suoritinaikaa, sovelluksen käytön aikana siirretyn tiedon määrää, käytetyn muistin määrää, massamuistin käyttöä, kellonaikaa ja sovelluksen käyntiaikaa (Andersson 1995, s. 19). Malli muodostetaan laskemalla jakauma kullekin mittarille ja tallettamalla tällainen tilastollinen profiili todennäköisyyksinä. Esimerkiksi voitaisiin tarkkailla kuukauden ajan UNIXin sendmail-ohjelman prosessin aiheuttamaa suorittimen kuormitusta ja laskea sen jälkeen, millä välillä suorittimen kuormitus tämän prosessin osalta tulisi normaalisti olla.

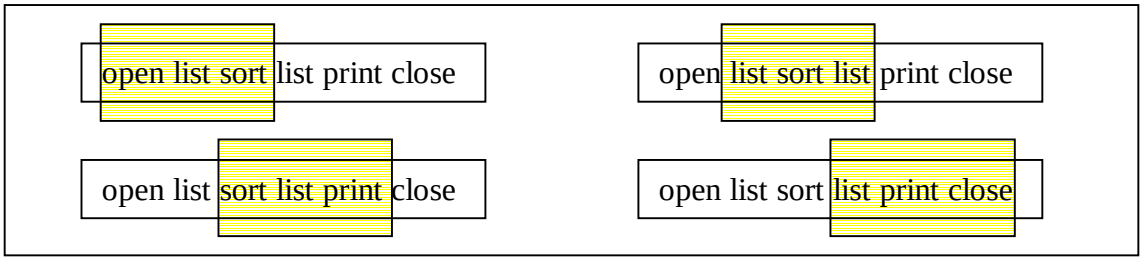
Tilastollisesti kuvaavien mallien käsittely on kuitenkin laskennallisesti raskasta, ja varsinkin jos tilastolliset metriikat kytketään ajalliseen järjestykseen, voi mallista tulla helposti laskennallisesti käyttökelvoton. Denningin (1987) yleinen malli on kuitenkin

kokonaisuutena niin monipuolinen, että sen varaan voi helposti rakentaa muuta tarkastelua. Tätä mallia onkin käytetty useiden tunkeutujantunnistamismallien pohjana.

Hofmeyer, Forrest ja Somayaji (1998, s. 3) ovat ottaneet täysin vastakkaisen näkökulman ja yrittävät mallintaa järjestelmän normaalia toimintaa mahdollisimman yksinkertaisesti. Hofmeyerin ym. (1998) mallissa yksinkertaistaminen on viety jopa niin pitkälle, että käyttäjä on poistettu mallista ja tekijänä ovat järjestelmäprosessit (privileged processes). Profiilit rakennetaan tässä mallissa siis prosesseille eikä käyttäjille. Kunkin prosessin esittämästä järjestelmäkutsujen sarjasta rakennetaan p-mittaisten tasapituisten ajallisesti järjestyksessä olevien jaksojen tietokanta.

Prosessin järjestelmäkutsut jaksotetaan siten, että kussakin tietueessa on p kappaletta järjestelmäkutsuja ja p-pituista ikkunaa liu'utetaan järjestelmäkutsujen sarjan yli siten, että saadaan kaikki yksilölliset ajallisesti järjestyksessä olevien järjestelmäkutsujen p-mittaiset jaksot. Esimerkiksi ikkunan koolla 1 saataisiin vain kaikki yksilölliset järjestelmäkutsut. Ikkunan koolla 2 puolestaan saataisiin järjestelmäkutsujen peräkkäisiä pareja. Ikkunan koolla on siis vaikutusta siihen, miten hyvin järjestelmä pystyy tunnistamaan järjestelmän normaalikäyttäytymistä. Liian pieni ikkuna ei kuvaa käyttäytymistä ja liian suuri ikkuna aiheuttaa liikaa vääriä hälytyksiä, sillä järjestelmäkutsujen sarja sellaisenaan ei toistu.

Esimerkiksi järjestelmäkutsujen sarjasta "open list sort list print close" saataisiin ikkunan pituudella p = 3 jaksot "open list sort", "list sort list", "sort list print", "list print close" (KUVIO 5). Toisaalta ikkunan pituudella p = 1 saataisiin vain yksittäiset järjestelmäkutsut "open", "list", "sort", "print" ja "close", jolloin malli ei kuvaisi käyttäytymistä, vaan ainoastaan käytettyjä komentoja.



KUVIO 5. Jaksojen muodostaminen. Hofmeyerin ym. (1998) ja Forrestin ym. (1996) malleissa tallennetaan järjestelmäkutsujen ajallisesti järjestettyjä vakiomittaisia jaksoja liu'uttamalla ikkunaa jäljitettyjen kutsujen sarjan yli.

Tekemissään empiirisissä kokeissa sendmail-, lpr- ja ftpd-prosessien datalla Hofmeyer ym. (1998, s. 10) päätyivät jakson pituuteen p = 10 ja rakensivat merkittävän pieniä tietokantoja, sendmailin tapauksessa vain 9085 tavua. Jaksot tallennettiin puurakenteeksi, jossa jokin tietty jakson aloittava järjestelmäkutsu oli puun juuri ja sen seuraajat aina seuraavia solmuja. Hofmeyerin (1998) malli on mielenkiintoinen yksinkertaisuutensa vuoksi, vaikka tutkijat myöntävätkin, että yksinkertainen malli ei ehkä ole sellaisenaan vielä riittävä. Malli käsittelee toimintoja sekä niiden ajallista järjestystä, eli Denningin (1987) mallin kaksi olennaista kohtaa poikkeamien tunnistuksessa on huomioitu.

Lane (1998) on rakentanut tunkeutujan tunnistamisen mallinsa hyvin samanlaisella periaatteella kuin Hofmeyer ym. (1998). Malli on kuitenkin monipuolisempi ja ottaa huomioon enemmän ominaisuuksia normaalikäyttöä mallinnettaessa. Lisäksi mallin sisäinen esitys ei ole puu, vaan se koostuu esimerkeistä instanssiperusteisen oppimistavan (instance based learning) mukaan (Lane 1998, s. 6). Malli muodostaa profiileja käyttäjäkohtaisesti, eli kukin rakennettu profiili koskee yhtä käyttäjää. Profiili rakennetaan UNIX-käyttöliittymän komentojen, niiden valitsimien ja viitattujen tiedostojen määrien ajallisesti järjestetyistä vakiomittaisista jaksoista. Lanen mukaan mallia voisi kuitenkin soveltaa mihin tahansa komentojen sarjaan. (Lane 1998, s. 28)

Lanen (1998) tutkimusta ovat kuvanneet järjestelmän toiminnan näkökulmasta laajemmin Lane ja Brodley (1999). Lanen ja Brodleyn (1999, s. 303) kuvaamassa mallissa profiili muodostetaan käyttöliittymäkomentojen sarjasta, jossa kohteet on korvattu niiden vastaavilla määrillä. Malli siis sisältää tekijän ja toiminnan kuvauksen, mutta varsinaista kohdetta ei mallinneta. Komennot ja niiden valitsimet kuvaavat kirjoittajien mielestä parhaiten käyttäjän käyttäytymistä valitussa ympäristössä. On kuitenkin huomattava, että toisenlaisessa ympäristössä myös toiminnan kohteella voi olla tärkeä merkitys.

Käyttöliittymäkomentojen sarja ositetaan samaan tapaan kuin edellä kuvatussa Hofmeyerin (1998) mallissa. Jokainen välilyönnein erotettu osa (token) on p-pituisen jakson aloituskohta, ja ajallisesti järjestyksessä olevia yksilöllisiä ja p-pituisia jaksoja erotetaan sarjasta liu’uttamalla p-mittaista ikkunaa sarjan alusta loppuun. Esimerkiksi

käyttäjän sarja ”cat osa1.txt osa2.txt osa3.txt > koko.txt ls –l | more” muuntuisi ensin muotoon ”cat <3> > <1> ls –l | more”, kun tiedostonimet korvataan niiden määrillä. Jos ikkunan mitta olisi 4, saataisiin sarjasta jaksot ”cat <3> > <1>”, ”<3> > <1> ls”, ”> <1> ls –l” ja ”ls –l | more”. Nämä yksilölliset jaksot tallennetaan opetusvaiheessa käyttäjän profiiliksi. Lane ja Brodley (1999, s. 303).

Tekemissään empiirisissä kokeissa Lane ja Brodley (1999, s. 302) havaitsivat tunnistamisen kannalta optimaalisen komentojen jakson pituuden p riippuvan käyttäjästä. Testidatan perusteella Lane ja Brodley toteavat, että muodostettaessa profiilia kyseisessä ympäristössä jakson pituus p = 10 oli sopiva kompromissi kaikkien testattujen käyttäjien kesken. Mielenkiintoinen yksityiskohta on, että tämä havaittu pituus vastaa Hofmeyer ym. (1998) saamia tuloksia.

Lanen (1998) menetelmä profiilin rakentamiseksi on yksinkertaisuutensa vuoksi hyvä lähtökohta käyttäjän käyttäytymisen mallintamiseen annettujen komentojen ajallisen järjestyksen perusteella. Käyttäjäprofiilin muodostaminen lyhyistä ajallisesti järjestetyistä komentojen jaksoista näyttää lupaavalta lähestymistavalta käyttäjän käyttäytymisen kuvaamiseksi.

Edellä kuvatun kaltainen malli, jossa käyttäjän profiili opitaan esimerkeistä ja tallennetaan instansseina mahdollistaa Lanen (1998, s. 11) mukaan jatkuvan oppimisen (continual learning) ja normaalitoiminnan muutoksen (concept drift) ajan mittaan. Tällaisen instanssiperusteisen profiilin päivittäminen on yksinkertaisempaa kuin ohjelmoidun tai tilastollisesti lasketun mallin muuttaminen. Koska käyttäjän toiminta kuitenkin todennäköisesti muuttuu, eikä mikään kerran opittu profiili voi sisältää kaikkea tarvittavaa tietoa, järjestelmä sisältääkin palautemekanismin, joka voi dynaamisesti päivittää profiilia tarpeen mukaan (Lane ja Brodley 1999, s. 300).

Normaalikäytön malli ja sitä käyttäen rakennetut normaalitoimintaa kuvaavat profiilit ovat pohjana vertailulle, kun käyttäjän toiminnasta yritetään etsiä mahdollisia väärinkäyttötapauksia. Väärinkäytön tunnistamisen tehokkuus riippuu paljolti käytetystä mallista ja siihen valituista normaalitoimintaa kuvaavista ominaisuuksista. Tämän vuoksi onkin syytä tarkoin harkita ja koestaa malliin valittavia ominaisuuksia.

Oleellista profiilien rakentamisen kannalta on päättää, mitä halutaan mallintaa. Mikäli mallinnetaan vain käyttäjän toimintaa, riittää hyvin tiivistetty esitys käyttäjän suorittamista toiminnoista ja niiden kohteista. Pelkkää käyttäjän normaalitoimintaa mallinnettaessa halutaan tietää mitä käyttäjä normaalisti tekee. Tunnistamisella pyritään varmistamaan, ettei käyttäjän tunnuksilla tehdä normaalista poikkeavia toimintoja. Mikäli halutaan mallintaa tapaa, jolla käyttäjä toiminnot suorittaa, täytyy profiiliin tallentaa enemmän tietoa eli pyrkiä tavalla tai toisella mittaamaan käyttäjän käyttäytymistä. Normaalia käyttäytymistä mallinnettaessa halutaan tietää, miten käyttäjä normaalisti toimii. Käyttäytymisen tunnistamisen perusteella pyritään varmistamaan, että käyttäjä on todella se, joka väittää olevansa. Käyttäytymisen tunnistamiseen voidaan käyttää muun muassa biometrisia suureita, kuten esimerkiksi näppäilynopeutta. Myös järjestelmän resursseja voidaan mitata käyttäytymisen tallentamiseksi ja tunnistamiseksi. Käyttäytymistä kuvaavien tilastollisten suureiden laskemisen hyöty täytyy kussakin järjestelmässä todeta empiirisesti, sillä niiden yhteys todelliseen käyttöön ei ole välttämättä intuitiivisesti perusteltavissa. Tämän lisäksi tilastollista laskentaa varten raakadataa täytyy tallentaa huomattavasti enemmän ja mallista tulee laskennallisesti raskaampi.

Käyttäjän käyttäytymisen mallintamisesta ajallisesti järjestettyinä komentojen jaksoina on saatu lupaavia tuloksia (Lane ja Brodley 1999). Erityisen hyvä ominaisuus tässä lähestymistavassa on se, ettei edellä mainittujen tekijän, toiminnan ja kohteen lisäksi tarvitse tallentaa muuta erillistä tietoa. Komentojen tallentaminen ajallisesti järjestettyinä lyhyinä jaksoina riittää. Lanen (1998) esittämä instanssiperusteinen oppimismalli ei välttämättä ole laskennallisesti yhtä tehokas kuin esimerkiksi puurakenteiset mallit, mutta koska varsinaista kiinteää normaalitoiminnan mallia ei rakenneta, on tunnistamisen perusteena olevan profiilin päivittäminen helppoa.

Käyttäjän normaalista toiminnasta ja käyttäytymisestä rakennetut profiilit toimivat väärinkäytön tunnistamisen perustana. Profiilien malli ja sen mukaan tallennettu tieto määrää pitkälti sen, millä tavoin järjestelmän tapahtumia voidaan analysoida ja poikkeamia normaalikäytöstä tunnistaa. Seuraavassa kohdassa on tarkasteltu tapoja, joilla voidaan tunnistaa poikkeamia vertaamalla järjestelmän uusia tapahtumia normaalitoiminnan tai -käyttäytymisen profiiliin.

2.5 Poikkeamien tunnistaminen muodostettujen profiilien perusteella

Poikkeamien tunnistaminen normaalikäytön profiilin perusteella riippuu mallista, jonka mukaan profiili on rakennettu. Käytetyimpiä malleja nykyisissä poikkeamaperusteisissa järjestelmissä tai niiden rakentamista kuvaavissa ehdotuksissa ovat tilastolliset menetelmät sekä tulevaa toimintaa tallennettujen mallien perusteella ennustamaan pyrkivät järjestelmät, kuten Hofmeyer ym. (1998) ja Lane (1998) suunnittelemat mallit. Perusajatukseltaan useat näistä tunnistamisen lähestymistavoista ovat, ainakin löyhästi, lähtöisin Denningin (1987) esittämistä ajatuksista (Lane 1998, s. 23).

Käytetyimpiä tilastollisia malleja ovat Denningin (1987, s. 6) ehdottamat operationaalinen malli, keskiarvo- ja keskihajontamalli sekä monimuuttuja-analyysi (Lane 1998, s. 23). Operationaalisessa mallissa aiemmin havaitusta normaalikäytöstä johdetaan kiinteät raja-arvot sille, mitä arvoja kukin mittari voi saada. Mittari voi olla esimerkiksi levynkäyttö tai tulostettujen sivujen määrä. Jos raja-arvo ylittyy, toiminta on epäilyttävää. Keskiarvo- ja keskihajontamallissa toiminta on epäilyttävää, jos toiminta jonkin yksittäisen mittarin suhteen on luottamusvälin ulkopuolella. Aiemmista normaalikäytön havainnoista lasketaan keskiarvo ja keskihajonta, jolloin saadaan tietyllä todennäköisyydellä luottamusväli, jonka sisällä uusien havaintojen arvot voivat vaihdella. Mallia voisi myös painottaa esimerkiksi uusimpien havaintojen suuntaan. Monimuuttuja-analyysissä mittarien luottamusvälit lasketaan keskiarvoon ja keskihajontaan perustuen, mutta yksittäisten muuttujien sijasta muuttujia vertaillaan yhdistelminä. Muuttujat voivat olla toisiinsa suoraan tai kääntäen verrannollisia. Denning (1987, s. 6)

Hofmeyerin ym. (1998, s. 7) mallissa väärinkäytön tunnistaminen tehdään vertaamalla tallennettuja järjestelmäkutsujen jaksoja järjestelmän nykyisestä toiminnasta muodostettuihin samanmittaisiin jaksoihin. Kaikkia uusia kutsujen jaksoja verrataan kaikkiin tallennettuihin kutsujen jaksoihin. Järjestelmäkutsujen jaksot on tallennettu prosesseittain puurakenteisiin tietokantoihin. Yksi esitetty vaihtoehto on yksinkertaisesti tutkia, löytyykö uusi kutsujen jakso prosessin tietokannasta. Mikäli jaksoa ei löydy, se merkitään epäilyttäväksi. Poikkeaman vahvuus voidaan laskea epäilyttäväksi merkittyjen jaksojen määränä tai niiden osuutena kaikista havaituista uusista jaksoista. Tämän ratkaisun ongelmana on se, että poikkeaman vahvuus riippuu havaitun sarjan

pituudesta, joka voi joillekin prosesseille olla teoriassa päättymätön. (Hofmeyer ym. 1998, s. 7)

Toinen tapa on laskea, kuinka paljon vertailtava havaittu jakso eroaa prosessin tietokannassa olevista jaksoista. Tällainen tarkastelu on paikallinen, eikä riipu siitä, kuinka pitkää sarjaa vertaillaan. Otetaan esimerkiksi historiallisten järjestelmäkutsujen sarja ”open list sort list print close” (TAULUKKO 1). Ikkunan pituudella p = 3 profiiliin tallennettuja jaksoja ”open list sort”, ”list sort list”, ”sort list print”, ”list print close” verrataan uuteen havaittuun sarjaan ”open list sort list list print close”. Havainnosta saadaan vastaavalla ikkunan pituudella jaksot ”open list sort”, ”list sort list”, ”sort list list”, ”list list print”, ”list print close”.

TAULUKKO 1. Järjestelmäkutsujen jaksojen eroavaisuuden laskenta. Tallennettujen järjestelmäkutsujen vakiomittaisia jaksoja sekä uusia havaittuja jaksoja ja niiden eroavaisuudet.

uudet havaitut	lähin vastaava tallennettu	eroavaisuus
open list sort	open list sort	0
list sort list	list sort list	0
sort list list	sort list print	1
list list print	sort list print	1
list print close	list print close	0

Ero lasketaan lähimpänä eroavaa uutta jaksoa olevan historiallisen jakson ja uuden jakson eroavaisuutena. Esimerkiksi uutta jaksoa ”sort list list” verrataan jaksoon ”sort list print” ja erovaisuudeksi saadaan 1, sillä jakson yksi osa on erilainen. Käytännössä tietysti kaikkien historiallisten jaksojen eroavaisuudet uusiin lasketaan ja pienimmät niistä otetaan mukaan tarkasteluun. (Hofmeyer 1998, s. 7)

Tehtiinpä tarkastelu kummalla tavalla tahansa, epäilyttäviksi laskettavien jaksojen määrälle tai niiden suhteelliselle osuudelle asetetaan tietty raja-arvo, jonka ylittymisen jälkeen voidaan laukaista hälytys ja ilmoittaa mahdollisesta väärinkäytöstä. Tunnistami-

nen on edellä esitettyyn tilastolliseen malliin verrattuna erittäin yksinkertaista ja mallin ajattelutapa on helppo hyväksyä.

Lanen ja Brodleyn (1999) esittämä malli on perusajatukseltaan hyvin samankaltainen Hofmeyerin (1998) mallin kanssa. Normaalitoiminnan profiilit ovat kuitenkin monipuolisempia, sillä mallinnettava toiminta koskee käyttäjän antamia komentoja, eikä järjestelmän prosessien tekemiä järjestelmäkutsuja. Tallennettujen jaksojen vertailu uusiin jaksoihin tehdään samoin kuin Hofmeyerin mallissakin, mutta profiilit ovat käyttäjäkohtaisia. Kunkin käyttäjän antamista komennoista muodostettuja jaksoja verrataan opetusvaiheessa tallennettuihin kyseisen käyttäjän jaksoihin.

Historiallisten ja käytönaikaisten jaksojen erilaisuuden vertailu määritellään Lanen ja Brodleyn (1999) mallissa vastaavuusmitan (similarity measure) kautta. Jakson sisällä peräkkäiset yhtenevät osat (token) lisäävät vastaavuutta enemmän kuin yksittäiset hajallaan sijaitsevat osat. Otetaan esimerkiksi kuuden osan jakso pituudeltaan $p = 6$. Ensimmäinen jakson yhtenevä osa saa painon 1, toinen yhtenevä osa painon 2 ja kolmas saa painon 3. Neljäs ja viides osa ovat erilaisia ja saavat molemmat painon 0. Kuudes osa on yhtenevä ja saa painon 1. Vastaavuusmitan arvoksi tulee painojen summa eli 1+2+3+0+0+1=7 (TAULUKKO 2). (Lane ja Brodley 1999, s. 302)

Suurin mahdollinen vastaavuuden arvo, jos kaikki historiallisen ja uuden jakson osat ovat yhteneviä, on 1+2+3+4+5+6=21 eli yleisesti $p(p+1)/2$. Uuden komentosarjan jakson vastaavuus kaikkiin profiilissa oleviin jaksoihin nähden on laskettujen vastaavuuksien maksimi. (Lane ja Brodley 1999, s. 302)

TAULUKKO 2. Uusi ja tallennettu komentojen jakso sekä painot yhteneville ja eroaville jakson osille. Peräkkäiset yhtenevät jakson osat kasvattavat painoa ja jos ne eroavat, paino palaa alkutasolleen.

	osa 1	osa 2	osa 3	osa 4	osa 5	osa 6
uusi	cat	<3>	>	<1>	ls	-l
tallennettu	cat	<3>	>	<2>	cat	-l
paino	1	2	3	0	0	1

Uusien ja tallennettujen jaksojen vertailusta saatua vastaavuusarvoa ei kuitenkaan käytetä suoraan määrittämään käyttäjän toimintaa epäilyttäväksi. Lane ja Brodley (1999, s. 303) laskevat keskiarvon tietylle määrälle jaksoja, eli raja-arvoa verrataan tällöin jaksojen vastaavuuksien summaan jaettuna niiden määrällä. Kokeissa sopivaksi määräksi katsottiin 100 jaksoa. Vastaavuudelle voidaan laskennan jälkeen asettaa raja, jonka alle jäävät jaksojen lasketut keskiarvot aiheuttavat hälytyksen. Lane ja Brodley 1999, s. 305) ovat tarkastelleet myös ylärajan asettamista liian samanlaisina toistuvien jaksojen toteamiseksi.

Varsinainen poikkeamien tunnistaminen perustuu käyttäjän normaalitoiminnasta aiemmin tallennetun tiedon ja siitä normaalitoiminnan mallin mukaan luodun profiilin käyttöön. Profiilia verrataan vastaavalla tavalla suodatettuun tietoon uudesta toiminnasta, ja mikäli toiminta ei ole tietyn raja-arvon tai luottamusvälin mukaan normaalia, sitä pidetään epäilyttävänä. On tärkeää tehdä ero käyttäjän toiminnan ja käyttäytymisen havainnoinnin välille, sillä niiden mallintaminen ja tunnistaminen eroavat toisistaan merkittävästi, vaikka profiilien rakentamiseen voitaisiinkin käyttää samaa tietoa.

Edellä on kuvattu poikkeamaperusteisen tunnistamisen yleisiä periaatteita sellaisina kuin tutkijat ovat niitä aiemmin esittäneet. Tutkijat usein esittävät omia mallejaan niihin uskoen ja melko kritiikittömästi. Poikkeamaperusteisella tunnistuksella ja kullakin erityisellä mallilla on kuitenkin vahvuuksia ja heikkouksia, joita käsitellään seuraavassa luvussa.

3 POIKKEAMAPERUSTEISEN VÄÄRINKÄYTÖNTUNNISTUKSEN VAHVUUKSIA JA HEIKKOUKSIA

Väärinkäytön- ja tunkeutujantunnistamisen tutkimuksessa ei vielä nykyisellään ole löydetty yhtä hyvää periaatetta rakentaa sopivia järjestelmiä. Esimerkiksi Allen ym. (2000, s. 17) pitävät tunkeutujantunnistamisteknologiaa vielä kypsymättömänä ja tutkimusalaa dynaamisena. Molemmista tunnistamisen pääperiaatteista, sekä poikkeama- että tunnisteperusteisesta tunnistamisesta, on löydettävissä heikkouksia ja vahvuuksia. Tunkeutujan tunnistamisjärjestelmien ominaisuuksia taksonomiassaan laajasti listannut Axelsson (2000, s. 2) näkee tunnisteperusteisen lähestymistavan pääasialliseksi ongelmaksi sen, ettei kaikkia mahdollisia hyökkäysmalleja voida tallentaa tunnisteina etukäteen, ja järjestelmä ei tällöin tunnista uudenlaisia hyökkäyksiä. Vastaavasti ongelmallisena poikkeamaperusteisten menetelmien kannalta Axelsson (2000, s. 2) näkee niiden aiheuttamien väärien hälytysten suuren määrän. Pääperiaatteiden heikkoudet ja vahvuudet risteävät siten, että toisen heikkous on toisen vahvuus. Tällöin yhtä periaatetta tarkasteltaessa on otettava vertailun vuoksi usein mukaan myös toinen.

Allen ym. (2000, s. 11) on vertaillut pääperiaatteiden vahvuuksia ja heikkouksia neljän ominaisuuden perusteella: luokittelutiedon tyypin, konfiguroinnin helppouden, raportoinnin kattavuuden ja raportoinnin tarkkuuden mukaan. Luokittelutiedon tyyppi vastaa Axelssoninkin (2000, s. 2) tunnistamaa periaatteellista eroa, jolla jaoteltuna tunnisteperusteisilla järjestelmillä on tieto mahdollisista hyökkäyksistä ja poikkeamaperusteisilla järjestelmillä normaalitoiminnasta. Konfiguroinnin helppous kuvaa sitä työmäärää, joka tarvitaan joko hyökkäysten tunnisteiden ohjelmoimiseen tai normaalitoiminnan mallin luomiseen. Raportoinnin kattavuus tarkoittaa mahdollisesta havaitusta hyökkäyksestä tällaisella järjestelmällä saatavien tarkempien informatiivisten tietojen määrää ja raportoinnin tarkkuus edellä kuvattuja väärien negatiivisten ja väärien positiivisten ongelman häiritsevyyttä järjestelmässä. (Allen ym. 2000, s. 11) Yksinkertaistaen ominaisuuksien vertailu voidaan tiivistää kahteen ryhmään: järjestelmän rakentamisvaiheeseen sekä varsinaiseen poikkeamien tunnistamiseen liittyviin.

Lupaavimpana lähestymistapana Axelsson (2000, s. 7) näkee yhdistelmätunnistamisen, sillä malli voi ainakin teoriassa käyttää hyväkseen tietoa sekä tunnetuista hyökkäyksistä että käyttäjän tai prosessien aikaisemmasta normaalitoiminnasta. Koska tämä tutkielma keskittyy poikkeamaperusteiseen väärinkäytön tunnistamiseen, on seuraavassa käsitelty kuitenkin erityisesti poikkeamaperusteisiin menetelmiin liittyviä heikkouksia ja vahvuuksia edellä esitettyä Allenin (2000, s. 11) jaottelua soveltaen.

3.1 Vahvuudet ja heikkoudet normaalikäytön mallia muodostettaessa

Poikkeamaperusteisen järjestelmän selvä etu tunnisteperusteiseen järjestelmään verrattuna on se, ettei epäilyttävän toiminnan tunnistamista varten tarvita tietoa mahdollisesti suoritettavasta hyökkäyksestä. Tämä vastaa Allenin (2000, s. 11) jaottelun mukaista luokittelutiedon tyyppiä, joka poikkeamaperusteisessa järjestelmässä tarkoittaa sitä, että järjestelmään on jollain tavoin tallennettu malli normaalista käytöstä. Järjestelmä pitää kaikkea muuta kuin mallinnettua käyttöä poikkeavana. Mikäli järjestelmä oppii opetusvaiheessa vain todellista normaalikäyttöä, on järjestelmä luotettava kaiken poikkeavan toiminnan tunnistamisen suhteen. Tietoturvan kannalta tällainen oletuksena kaikkea epäilyttävänä pitävä järjestelmä on perusajatukseltaan luotettavampi kuin oletuksena kaiken salliva tunnisteperusteinen järjestelmä. Axelsson (1998, s. 5) toteaakin aiemman tutkimuksen tuloksena havaitun, että poikkeamaperusteiset järjestelmät pystyvät tunnistamaan myös vanhoista hyökkäystavoista muunnetut ja täysin uudet hyökkäykset juuri tämän periaatteellisen eron vuoksi. Vastaavasti tunnisteperusteinen järjestelmä ei pysty tunnistamaan kuin ne hyökkäykset, jotka siihen on ennalta mallinnettu. Tämä periaatteellinen ero vaikuttaa yhtä hyvin normaalikäytön mallin muodostamiseen kuin varsinaiseen tunnistusvaiheeseenkin.

Mallin muodostamisvaiheessa merkittävä etu Axelssonin (1998, s. 5) mukaan on se, että poikkeamaperusteinen malli voidaan rakentaa antamalla järjestelmän itse oppia normaalikäyttö. Järjestelmästä vastaavan henkilön ei tarvitse konfiguroida sitä käsin, ja järjestelmä voidaan periaatteessa jättää oppimaan normaalikäytön malli ilman valvontaa (Axelsson 1998, s. 5). Tämä ominaisuus vastaa Allenin (2000, s. 11) vertailuperusteiden toista kohtaa. Tutkijat eivät kuitenkaan ole yhtä mieltä konfiguroinnin helppoudesta. Allenin (2000, s. 11) mukaan poikkeamaperusteinen järjestelmä on vaikeampi

konfiguroida kuin tunnisteperusteinen järjestelmä, jos konfigurointi tehdään käsin. Järjestelmää varten tarvitaan perusteellinen tieto nykyisestä sekä odotetusta tulevasta normaalista käytöstä. Tämän lisäksi rakennettua mallia on myös ylläpidettävä. Automaattisten välineiden kehittäminen puolestaan vie aikaa ja niille syötettävän raakatiedon tulee olla ristiriidatonta. (Allen 2000, s. 11) Axelsson (2000, s. 5) puolestaan pitää hyökkäysten tunnisteiden kehittämistä aikaavievänä ja hankalana tehtävänä, johon pystyvät vain asiantuntijat. Voidaankin ajatella, että jonkin organisaation sisällä tieto käyttäjien normaalikäytöstä on yleensä helpommin saatavilla kuin tieto koko ajan kehittyvistä hyökkäysmenetelmistä. Jos oletetaan, että poikkeamaperusteisen järjestelmän profiilien rakentamiseen on käytettävissä automatisoitu järjestelmä, on mallin muodostaminen yksinkertaisempaa kuin tunnisteperusteisessa järjestelmässä. Käyttäjä-, järjestelmä- tai prosessikohtainen normaalitoiminnan profiili pystytään ideaaliympäristössä rakentamaan automaattisesti normaalitoiminnan periaatteellista mallia ja käyttäjän toimista kerättyä tietoa hyväksikäyttäen, eikä käsityötä tarvita.

Axelssonin (2000, s. 11) kuvaamaa täysin itsenäisesti oppivan järjestelmän ideaalia uhkaa valitettavasti väärinkäytön mahdollisuus jo opetusvaiheessa. Normaalikäytön profiilien rakentaminen eli väärinkäytöntunnistamisjärjestelmän opetusvaihe on Lanen (1998, s. 26) mukaan erittäin kriittinen vaihe. Pahimmassa tapauksessa järjestelmä oppii jo tässä vaiheessa pitämään jotakin luvatonta käyttöä täysin normaalina. Poikkeamaperusteisten järjestelmien ongelmia rakentamansa prototyyppijärjestelmän avulla tarkastelleet Lundin ja Jonsson (1999, s. 13) toteavat, että tunkeutujantunnistamisjärjestelmän olemassaolosta tietoinen käyttäjä voi helposti harhauttaa järjestelmää. Lane (1998, s. 26) mainitsee yhdeksi ratkaisumahdollisuudeksi rakentaa käsin väljät alustavat profiilit opetusvaiheen ajaksi. Toisaalta oppivan poikkeamaperusteisen tunnistuksen perusajatus on välttää käsin tehtävä määrittelytyö ja sen epätarkkuudet, joten tällainen ratkaisu vaikuttaa perin kummalliselta. Poikkeamaperusteisten järjestelmien malleissa onkin tehty usein perusoletus, jonka mukaan opetusvaiheessa käsiteltävä tieto ei sisällä väärinkäyttöyrityksiä, jolloin malli voidaan rakentaa sen perusteella. Tämä voidaan saavuttaa esimerkiksi siten, että järjestelmän turvallisuudesta vastaava asiantuntija valvoo uusia käyttäjiä tarkemmin manuaalisesti. Tämä tarkoittaa sitä, ettei automaattisestikaan oppivan järjestelmän tapauksessa voida täysin välttyä järjestelmän tietoturvasta vastaavien henkilöiden tekemältä käsityöltä.

Denningin (1987, s. 11) mielestä uutta käyttäjäprofiilia luotaessa se voitaisiin aluksi sitoa johonkin toiseen jo olemassa olevaan profiiliin tai profiilien ryhmään, jonka normaalikäytön malli olisi väljempi. Tällä tavoin voitaisiin varmistaa, ettei uuteen profiiliin tallennettava käyttäytymismalli sisällä järjestelmän kannalta täysin poikkeavaa toimintaa. Lundin ja Jonsson (1999, s. 13) ovat käyttäneet prototyypissään juuri tällaista periaatetta. Eräs mielenkiintoinen tapa voisi olla myös järjestelmän käyttöoikeusmääritysten käyttäminen alustavana profiilina. Mikäli järjestelmään on määritelty jonkinlaiset käyttöoikeudet, ei väärinkäytöntunnistamisjärjestelmän tulisi hyväksyä niiden määrittelemää laajempaa käyttöä. Denningin (1987, s. 5) mallissa tarkkaillaan myös järjestelmän palauttamia poikkeuksia. Käyttöoikeuksienvalvontajärjestelmän tai tiedostojärjestelmän palauttamia poikkeuksia voitaisiin periaatteessa juuri tällä tavoin hyödyntää profiileja rakennettaessa.

Lundin ja Jonsson (1999, s. 10) mainitsevat profiilien muodostamisvaiheen ongelmina edellä käsiteltyjen lisäksi jäljitettävien käyttäjien yksityisyyteen, resurssikäyttöön ja rajoittuneeseen mallinnukseen liittyvät vaikeudet. Opetusvaiheessa käyttäjien toiminnasta laajasti tietoja keräävä ja tallentava järjestelmä saattaa vaarantaa käyttäjien oikeuden yksityisyyteen. Järjestelmä voi tallentaa täysin vilpittömästäkin käytöstä sellaista tietoa, jonka käyttäjä ei haluaisi tulevan muiden tietoon. (Lundin ja Jonsson 1999, s. 10) Yksityisyyteen liittyvä näkökulma on merkittävä ja se on otettava huomioon poikkeamaperusteisia järjestelmiä rakennettaessa, sillä ongelma on sekä eettinen että joissain tapauksissa mahdollisesti myös juridinen.

Resurssikäyttöön liittyviin ongelmiin viitattiin jo edellisessä luvussa käsiteltäessä tilastolliseen malliin pohjautuvaa poikkeamaperusteista tunnistamista. Lane (1998, s. 10) toteaa, ettei tunnistusjärjestelmä saa kuluttaa valvottavan järjestelmän resursseja suoritinajan tai levytilan suhteen häiritsevästi. Poikkeamaperusteinen järjestelmä tallentaa kuitenkin yleensä enemmän tietoa mallin rakentamista varten kuin tunnisteperusteinen järjestelmä. Esimerkiksi Frankin (1994, s. 3) mukaan verkkoperusteisessa järjestelmässä jäljitystietoa muodostuu käyttäjää kohti tyypillisesti 3 - 35 megatavua työpäivän aikana. Poikkeamaperusteisissa järjestelmissä ja erityisesti tilastollista laskentaa hyödyntävissä malleissa jäljitystietojen käsittely ja niiden tallentaminen profiileiksi voi muodostua ongelmaksi valvottavan järjestelmän tehokkuuden kannalta,

koska tietoa käsitellään yhtälailla järjestelmää opetettaessa sekä varsinaista tunnistamista suoritettaessa. Kumar (1995, s. 8) onkin todennut, että tilastollisen poikkeamientunnistamisen ongelmana on monien mittarien käyttö, niiden jatkuva päivittäminen ja tämän vaatima suuri laskentateho.

Hofmeyerin ym. (1998, s. 10) tekemien kokeiden mukaan poikkeamien tunnistaminen ajallisesti järjestettyjen palvelimen komentosarjojen lyhyiden jaksojen perusteella vaatii vähemmän laskentakapasiteettia kuin tilastollisten mittarien käyttö. Periaatteiltaan yksinkertaisesta järjestelmästä saatujen hyvien tunnistustulosten lisäksi sen muodostamat normaalikäytön profiilit olivat verrattain hyvin pieniä, vain muutamia kilotavuja (Hofmeyer ym. 1998, s. 10). Lundinin ja Jonssonin (1999, s. 12) kokeissa verkkoperusteisella järjestelmällä jäljitystietoa muodostui noin yksi megatavu päivässä. Kokeet osoittivat, että tunnistamiseen käytetty aika oli lähes suoraan verrannollinen jäljitystiedon määrään ja yhden ominaisuuden lisääminen tunnistettavien joukkoon kasvatti jäljitystiedon määrää merkittävästi. (Lundin ja Jonsson 1999, s. 12) Poikkeamaperusteisen järjestelmän ongelmaksi saattaakin muodostua liian monimutkainen normaalikäytön malli, joka hidastaa tunnistamista ja varaa valvottavan järjestelmän resursseja.

Edellä käsitelty poikkeamaperusteisen tunnistamisen vahvuus, sen kyky havaita myös täysin uudet hyökkäysmallit, perustuu olettamukseen, että järjestelmästä tunnistamista varten kerättävät tiedot pystyvät kuvaamaan kyseisen väärinkäytön. Lundin ja Jonsson (1999, s. 13) toteavat, että esimerkiksi heidän prototyyppijärjestelmänsä profiilit eivät todennäköisesti pysty tunnistamaan kaikkia mahdollisia hyökkäyksiä. Tullakseen tunnistetuksi, väärinkäytön täytyy vaikuttaa niihin attribuutteihin, joiden perusteella tunnistaminen tehdään. (Lundin ja Jonsson 1999, s. 13) Jos tunnistamisjärjestelmää jäljittää esimerkiksi ohjelmien käyttöä käyttöjärjestelmätasolla, se ei pysty välttämättä tunnistamaan ohjelman sisällä tapahtuvaa väärinkäyttöä. Profiilien rajoittuneisuuden ongelma liittyy lähes kääntäen verrannollisesti edellä kuvattuun resurssikäytön ongelmaan. Resurssikäytön kannalta edulliset poikkeamaperusteisen järjestelmän profiilit saattavat olla kuvaamiskyvyltään rajoittuneita ja vastaavasti monipuoliset profiilit laskennallisesti käyttökelvottomia. Tämän vuoksi onkin syytä valita tarkoin kunkin ympäristön kannalta oleelliset jäljitystiedon ominaisuudet sekä resurssikäytön kannalta mahdollisimman edullinen mallintamisen tapa.

Poikkeamaperusteisen väärinkäytön tunnistamisen tärkeimpiä vahvuuksia järjestelmän opetusvaiheessa ovat mahdollisuus rakentaa normaalikäytön profiilit automaattisesti koneoppimisen menetelmiä käyttäen sekä tällaisten järjestelmien periaatteellinen kyky tunnistaa myös ennalta tuntemattomia hyökkäyksiä. Pääasiallisia heikkouksia profiilien rakentamisvaiheessa puolestaan ovat opettamisessa käytettävien tietojen luotettavuus, normaalikäytön mallin kuvaavuuden kattavuus sekä mallin tallentamiseen vaadittavat järjestelmäresurssit.

3.2 Vahvuudet ja heikkoudet tunnistamisvaiheessa

Allenin (2000, s. 11) jaottelun mukaan tunnistamisjärjestelmän toimivuutta tulee mitata sen teoreettisen kyvyn ja rakentamisen helppouden lisäksi myös sen raportoinnin kattavuuden ja tarkkuuden perusteella. Kuten edellisessä aliluvussa todettiin, poikkeamaperusteisen järjestelmän tunnistamisen periaate vaikuttaa myös raportoinnin tehokkuuteen. Kaikki käytettyjen mittarien mukaan aiemmin tuntematon toiminta katsotaan epäilyttäväksi, jolloin järjestelmä pystyy teoriassa tunnistamaan ja raportoimaan kaikki uudenlaisetkin hyökkäykset. Tämä periaatteellinen vahvuus on merkittävä, ja poikkeamaperusteisia järjestelmiä voidaankin pitää tietoturvan näkökulmasta luotettavina, koska ne eivät juurikaan tuota vääriä negatiivisia tunnistuksia.

Kumarin (1995, s. 8) mukaan merkittävä poikkeamaperusteisten väärinkäytöntunnistamisjärjestelmien heikkous on kuitenkin niiden aiheuttamien väärien hälytysten suuri määrä. Kun normaalikäytön profiilien perusteella halutaan varmuudella erotella väärinkäyttö normaalista käytöstä, asetetaan normaalikäytöstä poikkeamisen raja-arvot hyvin tiukasti. Tällöin pienet muutokset käyttäytymisessä aiheuttavat vääriä hälytyksiä, jotka työllistävät hälytyksiä tutkivia asiantuntijoita turhaan. (Kumar 1995, s. 8) Raportoinnin tarkkuutta ei voida pitää hyvänä, mikäli järjestelmä tuottaa paljon vääriä negatiivisia tai positiivisia tunnistuksia.

Kokeissaan Lundin ja Jonsson (1999, s. 12) havaitsivat prototyyppijärjestelmänsä muodostavan runsaasti vääriä hälytyksiä. Säännöllistä työtä tekevien käyttäjien kohdalla vääriä hälytyksiä tuli vähemmän, mutta vaihtelevasti työskentelevien kohdalla niitä

muodostui 0,5 – 1,2 käyttäjää kohti päivässä. Väärien hälytysten tarkastaminen kasvattaa turhaan järjestelmänvalvojien työtaakkaa ja siksi tutkijat ovatkin ehdottaneet, että järjestelmänvalvojien tarkastettaviksi voitaisiinkin antaa vain niiden käyttäjien toimet, joiden kohdalla poikkeamia esiintyy jotakin asetettua raja-arvoa enemmän. (Lundin ja Jonsson 1999, s. 12) Puhumattakaan perinteisemmistä tilastollisista malleista myös Hofmeyerin (1998) ja Lanen (1998) komentojen ajallista järjestystä hyödyntävät mallit kärsivät samasta väärien hälytysten aiheuttamasta ongelmasta. Lane (1998, s. 34) on pyrkinyt ongelmasta eroon käyttämällä tasoitettua vastaavuusmittaa (smoothed similarity measure). Historiallisten ja uusien jaksojen vertailussa lasketaan jaksojen epäilyttäväksi luokittelua varten keskiarvo pidemmästä jaksojen sarjasta, jolloin yhdellä pienen vastaavuuden jaksolla ei ole niin suurta merkitystä. Lane ja Brodley (1999, s. 303) kuitenkin huomauttavat, että tällaisen tunnistamisen lähestymistavan lisäksi tulisi käyttää tunnisteperusteisia tunkeutujantunnistamisjärjestelmiä havaitsemaan ennalta tunnettuja hyökkäysmalleja, sillä merkittävää vahinkoa voi saada aikaan tarkasteltua pienemmällä määrällä komentoja. Lane ja Brodley (1999, s. 303) pitävät parhaana vaihtoehtona yhdistelmätunnistamista, jossa kummankin päälähestymistavan parhaat puolet saadaan käyttöön.

Periaatteessa samaa ajattelutapaa kannattavat myös Hofmeyer ym. (1998, s. 9). Vaikka sekä väärien negatiivisten että väärien positiivisten luokitteluvirheiden minimointi olisi heidän mielestään toivottavaa, on tärkeämpää, etteivät järjestelmää rasita turhat väärät hälytykset. Tämän saavuttamiseksi voidaan kasvattaa tunnistamisen rajoja eli hyväksyä suurempia poikkeamia profiilista. Suurempien poikkeamien salliminen saattaa aiheuttaa myös joidenkin väärinkäyttötapausten hyväksymisen. Yksittäisten tunnistimien väärät negatiiviset eivät kuitenkaan haittaa, jos käytetään toisistaan riippumattomien tunnistimien kerroksittaista rakennetta. (Hofmeyer ym. 1998, s. 9) Käytännössä tämä tarkoittaa sitä, että järjestelmän suojaksi rakennetaan useita eri lähestymistapoihin, malleihin tai opetusaineistoihin perustuvia tunnistimia, jolloin tunkeutujan tai väärinkäyttäjän kiinnijäämisen todennäköisyys on suurempi.

Toinen Allenin (2000, s. 11) esittämistä tunnistamisvaiheeseen liittyvistä vertailuperusteista on raportoinnin laajuus. Allenin (2000, s. 11) mukaan poikkeamaperusteiset järjestelmät pystyvät yleensä tuottamaan hyökkäyksestä enemmän tietoa kuin

tunnisteperusteiset järjestelmät, jotka yleensä kertovat vain mikä hyökkäys on tapahtunut tai korkeintaan hyökkäyksen tunnistamiseen johtaneet tuntomerkit. Poikkeamaperusteiset järjestelmät kykenevät tarkempaan raportointiin, koska ne ilmoittavat kaiken mallinnettua normaalikäyttöä vastaamattoman käytön. (Allen 2000, s. 11) Raportoinnin laajuutta ja hälytyksen aiheuttaneiden tapahtumien tarkempien tietojen näyttämistä ei ole kirjallisuudessa juurikaan käsitelty, vaikka tämä ominaisuus on järjestelmien tietoturvasta vastaavien henkilöiden kannalta varmasti hyvin tärkeä. Raportoinnin kattavuus on kuitenkin selvästi poikkeamaperusteisen lähestymistavan vahvuus, sillä jo pelkästään mallien rakentamiseksi järjestelmästä kerätään yleensä tietoa mahdollisimman monella mittarilla.

Lane (1998, s. 24) pitää poikkeamaperusteisen lähestymistavan merkittävänä vahvuutena sitä, että tunkeutujantunnistamisen lisäksi samoilla periaatteilla voidaan tunnistaa myös organisaation valtuutettujen käyttäjien suorittamaa väärinkäyttöä. Suurin osa tunkeutujantunnistamisen tutkimuksesta on jo käsitteen määrittelynkin perusteella keskittynyt organisaation suojaamisen ulkoiselta vaaralta, vaikka yhtä merkittävää olisi tarkastella organisaation sisäisiä riskitekijöitä. Valtuutettujen käyttäjien suorittama väärinkäyttö on merkittävää, sillä tutkimusten mukaan suuri osa tietovahingoista, tutkitusta kohteesta riippuen 30 - 80 %, johtui organisaation omien työntekijöiden suorittamasta väärinkäytöstä. Tällainen sisäinen väärinkäyttö näkyy poikkeamaperusteiselle tunnistamisjärjestelmälle käyttäjän käyttäytymisen nopeana muuttumisena. Lane (1998, s. 24) Poikkeamaperusteisten järjestelmien edut organisaation sisäisen väärinkäytön tunnistamisessa ovat selvät, sillä automaattisesti muodostettavien normaalikäytön profiilien avulla on vähemmän työlästä tunnistaa esimerkiksi monimutkaisen sovelluksen poikkeava käyttö kuin luoda väärinkäytön tunnisteet jokaiselle mahdolliselle tilanteelle. On tosin huomattava, että joidenkin tietotasolla tapahtuvien ja ennalta määriteltävissä olevien väärinkäyttöyritysten tunnistamiseen tunnisteiden kaltaiset säännöt ovat todennäköisesti tehokkaampia.

Poikkeamaperusteisen väärinkäytön tunnistamisen merkittävä ongelma on väärien hälytysten suuri määrä, joka työllistää turhaan asiantuntijoita ja heikentää tunnistamisen uskottavuutta. Ongelmaa voidaan yrittää ratkaista käyttämällä valintaa käsintarkasteltaviksi esitettävien hälytysten suhteen sekä rakentamalla järjestelmän suojaksi useampia

raja-arvoiltaan väljempiä ja toisistaan riippumattomia tunnistajia. Toisaalta poikkeamaperusteisen tunnistamisen vahvuuksia ovat sen kyky tunnistaa myös aiemmin tuntemattomia hyökkäysmalleja sekä sen käytännöllisyys myös valtuutettujen käyttäjien suorittaman väärinkäytön tunnistamisessa. Poikkeamaperusteisen järjestelmän toimintakyvyn varmistamiseksi normaalikäytön profiilien tulee oppia koko ajan, jotta ne pystyvät sopeutumaan valtuutetun käyttäjän muuttuvaan toimintaan. Seuraavassa kohdassa on käsitelty jatkuvaan oppimiseen liittyviä ongelmia.

3.3 Järjestelmän kyky oppia ja mukautua valtuutetun käyttäjän toiminnan muutoksiin

Lanen (1998, s. 13) mukaan väärinkäytön tunnistamisessa käytettävän poikkeamaperusteisen mallin tulisi oppia kaiken aikaa uutta, sillä mikään kerran muodostettu opetusjoukko (training set) ei voi kuvata käyttäjän muuttuvaa käyttäytymistä. Lisäksi järjestelmän tulisi oppia itsenäisesti ilman ihmisen ohjausta, mutta se ei saisi kuitenkaan oppia haitallisia väärinkäyttömalleja (Lane 1998, s. 13). Tilanne on helpompi kuin järjestelmää ensimmäistä kertaa opetettaessa, sillä opetusjoukkoa päivitettäessä käyttäjän käyttäytymisestä on kuitenkin olemassa jo jotain tietoa, johon uusia tietoja voidaan verrata. Ongelmaksi kuitenkin muodostuu, miten paljon uusi opetettava toiminta tai käyttäytyminen voi poiketa aiemmin opitusta.

Lanen (1998, s. 58) mukaan käytännöllinen ratkaisu jatkuvaan oppimiseen voisi olla käyttäjän hitaasti muuttuvan käyttäytymisen tarkkailu ja tallennus. Jos esimerkiksi uutta komentoa käytetään yhdessä vanhojen tunnistettujen komentosarjojen kanssa käyttäytymismallin mukaan, eikä yhdistelmä vielä ylitä asettua epäilyttävän raja-arvoa, voitaisiin tällainen uusi komento opettaa järjestelmälle. Hitaat käyttäytymisen muutokset voitaisiin tallentaa käyttäjän profiiliin ja vain nopeat ja merkittävät ohjattaisiin ihmisen tarkastettaviksi. (Lane 1998, s. 58) Lundin ja Jonsson (1999, s. 13) pitävät tällä tavoin oppivan poikkeamaperusteisen järjestelmän periaatetta ongelmallisena, sillä järjestelmän olemassaolosta tietoinen vihamielinen käyttäjä voi käyttäytymistään hitaasti muuttamalla saada järjestelmän hyväksymään alunperin epäilyttävää käyttöä. Lanekin (1998, s. 25) toteaa, että väärinkäytöntunnistusjärjestelmän tunteva väärinkäyttäjä voisi halutessaan vähitellen opettaa järjestelmää hyväksymään

alkuperäisen mallin mukaan epäilyttävää käyttöä. Mahdollisena ratkaisuna tällaiseen ongelmaan Lane esittää käyttäjän toiminnan mallintamista useammilla tasoilla HAYSTACK-järjestelmän tapaan. Käyttäjän käyttäytymisen mallintava profiili voisi olla suppeampi ja mahdollistaa muutokset, kun taas useiden käyttäjien käyttäytymismalleista koostuva käyttäjäryhmäprofiili olisi laajempi ja staattinen. (Lane 1998, s. 25)

Käyttäjän muuttuvaan toimintaan mukautumaan pystyvä poikkeamaperusteinen väärinkäytöntunnistamisjärjestelmä saattaa pahimmillaan oppia vähitellen hyväksymään väärinkäytön normaalina toimintana. Varsinkin tilastollisia resurssikäyttöön perustuvia metriikoita käyttävien järjestelmien yhteydessä tämä on merkittävä ongelma. Ongelman ratkaisemiseksi järjestelmä voitaisiin oppimisen aikana sitoa erilaisiin alustaviin normaalikäytön malleihin, kuten kaikkia käyttäjiä kuvaavaan yhdistelmäprofiiliin. Kuitenkin vain oppivan järjestelmän avulla pystytään ratkaisemaan pahin poikkeamaperusteista väärinkäytön tunnistamista vaivaava ongelma eli väärät hälytykset, jotka johtuvat valtuutetun käyttäjän normaalikäytön muuttumisesta tai alunperin epätäydellisistä profiileista.

Poikkeamaperusteisilla väärinkäytöntunnistamisen malleilla on vahvoja ja heikkoja puolia. Yleisten rajoitusten tietäminen on tärkeää järjestelmiä suunniteltaessa tai niitä rakennettaessa. Ehdottomasti tärkeimmät vahvuudet ja samalla heikkoudet liittyvät tunnistamisen ehdottoman kieltävyyden periaatteeseen. Tällainen järjestelmä pystyy periaatteessa tunnistamaan myös kaikki uudet tavat väärinkäyttää järjestelmää, mutta toisaalta se rasittaa järjestelmänvalvojia liian usein väärillä hälytyksillä. Jokaisella sovellusalueella on lisäksi omia rajoituksiaan ja ominaispiirteitään. Seuraavassa luvussa käsitellään relaatiotietokannanhallintajärjestelmiin liittyviä poikkeamaperusteisen väärinkäytön tunnistamisen erityispiirteitä ja aiemman tutkimuksen esittämiä malleja.

4 VÄÄRINKÄYTÖN TUNNISTAMISEN ERITYISPIIRTEITÄ RELAATIOTIETOKANNANHALLINTAJÄRJESTELMISSÄ

Chungin ym. (1999, s. 1) mielestä organisaatioiden arvokkainta omaisuutta on tieto, jota pitää pystyä sekä suojelemaan että käsittelemään tehokkaasti. Tietokannanhallintajärjestelmät tarjoavat oikein käytettynä tehokkaan tavan toteuttaa molemmat vaatimukset. Käyttöoikeuksien määrittäminen käsin täysin aukottomasti on kuitenkin vaikeaa ja työlästä. Tutkijoiden relaatiotietokantojen käyttöoikeuksien hallintaa helpottamaan kehittämiä malleja, kuten Bertinon ym. (1999) mallia ei ole ainakaan vielä toteutettu käytännön kaupallisiin järjestelmiin. Tunkeutujantunnistamisjärjestelmät ovat helpottaneet tilannetta käyttöjärjestelmissä ja verkkoympäristöissä, joihin pääosa tutkimuksesta on keskittynyt. Verkon tai käyttöjärjestelmän tasolla toimiva tunkeutujan tai väärinkäytön tunnistamisjärjestelmä ei kuitenkaan pysty tunnistamaan tiedon ja tapahtumien semanttisia suhteita, koska ne tarkkailevat useimmiten vain käyttäjän antamia komentoja tai järjestelmäkutsuja. Yksityiskohtaisempi väärinkäytön tunnistaminen on tarpeen myös siksi, että valtuutettujen käyttäjien suorittama väärinkäyttö on yleistä. Pelkästään mahdollisten tunkeutujien tarkkailu ei tällöin riitä.

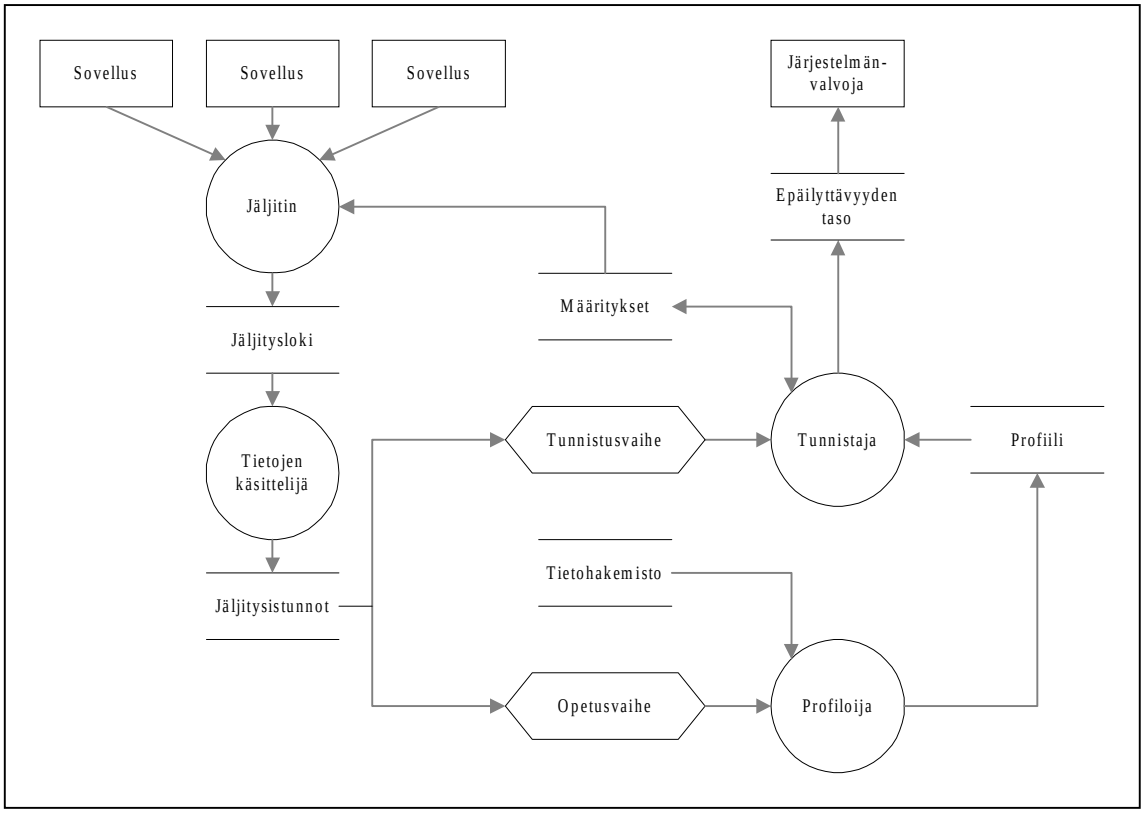
Nykyiset tietokannanhallintajärjestelmät mahdollistavat yhä enemmän myös sovelluslogiikan toteuttamista tietokannan tasolla. Yhä monimutkaisemmat rakenteet vaativat uudenlaista lähestymistapaa väärinkäytön tunnistamiseen. Jotta kuitenkin vältyttäisiin turhauttavalta käsityöltä, tulee pyrkiä löytämään optimaalinen taso tunnistamisen tarkkuuden suhteen. Poikkeamaperusteinen väärinkäytön tunnistamisen lähestymistapa on soveltuvin tietokannan monimutkaisen rakenteen vuoksi sekä valtuutettujen käyttäjien väärinkäytön tunnistamiseksi. Tunnisteperusteisella lähestymistavalla sääntöjä pitäisi luoda varsin paljon. Tietokannanhallintajärjestelmien yhteydessäkin nämä kaksi lähestymistapaa voivat kuitenkin täydentää toisiaan, kuten jäljempänä osoitetaan.

Seuraavassa on tarkasteltu relaatiotietokannanhallintajärjestelmien yhteyteen rakennettavien poikkeamaperusteisten väärinkäytöntunnistamisjärjestelmien erityispiirteitä yleisen rakenteen, jäljitystietojen keräämisen, käyttäjän toiminnan mallintamisen ja profiilin muodostamisen sekä väärinkäytöntunnistamisen osalta. Ai-

emmin erityisesti relaatiotietokannan väärinkäytöntunnistamista ovat käsitelleet Chung ym. (1998), Chung ym. (1999), Ingriswang ja Liu (2001) sekä Liu (2001).

4.1 Relaatiotietokannan poikkeamaperusteisen väärinkäytöntunnistamisjärjestelmän rakenne

Relaatiotietokannan yhteyteen rakennettavan poikkeamaperusteisen väärinkäytöntunnistamisjärjestelmän tulee sisältää osat ainakin jäljitystiedon keräämistä, profiilin muodostamista ja poikkeamien tunnistamista varten. Chungin ym. (1999, s. 4) DEMIDS-järjestelmässä (Detection of Misuse in Database Systems) jäljitin (auditor) kerää sovellusten tekemät SQL-kyselyt tietokannanhallintajärjestelmän omaa auditointijärjestelmää hyväksikäyttäen (KUVIO 6). Aiemmassa artikkelissaan Chung ym. (1998, s. 1) ovat todenneet järjestelmän sijoittuvan tietokantasovellusten ja tietokannanhallintajärjestelmän väliin. Toisaalta voitaisiin kuitenkin ajatella, että tietokannanhallintajärjestelmän sisällä tietoa varastoiva ja käsittelevä järjestelmä ei ole sovellusten ja hallintajärjestelmän välissä, vaan sovellusten rinnalla tietokannanhallinta-



KUVIO 6. Relaatiotietokannan poikkeamaperusteisen väärinkäytöntunnistamisjärjestelmän yleisrakenne Chung ym. (1999, s. 4) mukaan. Opetusvaiheessa sovelluksista tulevat jäljitystiedot kerätään jäljittimellä jäljityslokiin, josta tietojen käsittelijä muodostaa jäljitysistunnot. Profiloija muodostaa näistä edelleen profiilit. Tunnistusvaiheessa uusia jäljitysistuntoja ja profiileja verrataan keskenään.

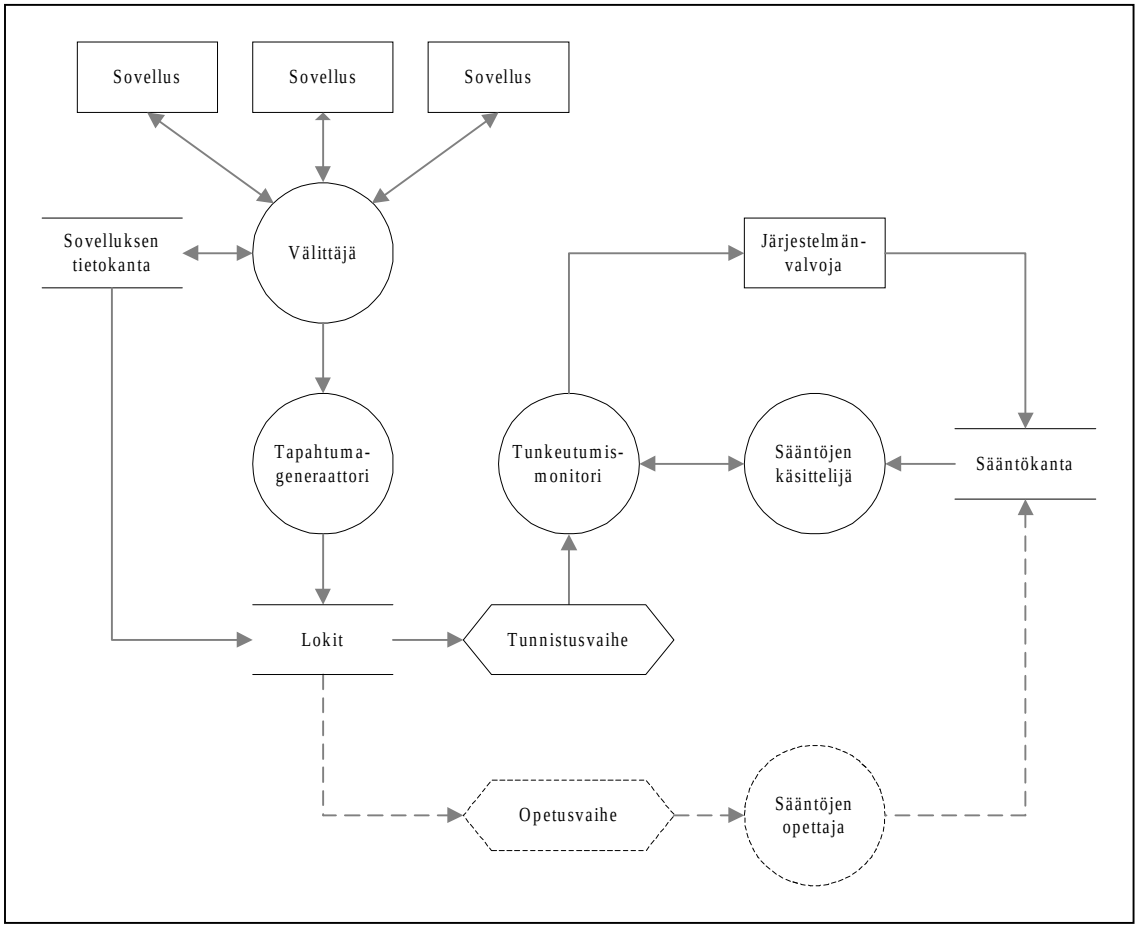
järjestelmässä. Jäljitystiedot tallennetaan jäljityslokiin (audit log) ja jalostetaan tietojenkäsittelijässä (data processor) edelleen ryhmitellyiksi jäljitysistunnoiksi (audit session). Tämän jälkeen profiloija (profiler) muodostaa opetusvaiheessa ryhmittelyn perusteella profiilit (profile), jotka voivat olla esimerkiksi käyttäjäkohtaisia. Profiloija käyttää hyväkseen myös tietohakemiston (data dictionary) kuvausta tietokannan rakenteesta. Tuotantokäytössä järjestelmän tunnistaja (detector) vertaa profiileja ja uusia jäljitysistuntoja toisiinsa. (Chung ym. 1999, s. 4)

Järjestelmän rakenne on perustasolla analoginen Lanen (1998) esittämän järjestelmän rakenteen kanssa. Järjestelmä oppii normaalikäytön automaattisesti käyttäjän aiemmista toimista koneoppimisen periaatteiden mukaan ja tunnistaa myöhemmin väärinkäytön poikkeamien perusteella.

Ingriswangin ja Liun (2001, s. 4) esittämä AAID-järjestelmän rakenteen kuvaus (KUVIO 7) on huomattavasti yleisemmällä tasolla. Järjestelmässä sovellusten ja tietokannan välillä oleva välittäjä (mediator) tallentaa SQL-kyselyt ja tekee niistä lokeja. Tapahtumageneraattori (event generator) lukee lokeja ja luo tapahtumia, jotka saattavat olla väärinkäyttöä, ja välittää ne edelleen tunkeutumismonitorille (intrusion monitor). Monitori tallettaa tapahtumat ja välittää ne sääntöjen käsittelijälle (rule processor). Tarvittaessa monitori myös tekee hälytyksen välittäjälle, joka voi reagoida tunkeutumisyrityksiin. Sääntöjen käsittelijä vertaa tapahtumia sääntökannassa (rule base) oleviin epänormaalia käyttöä kuvaaviin sääntöihin. Mikäli tapahtuma vastaa sääntöä, käsittelijä tekee hälytyksen. Kannassa olevat säännöt voidaan tehdä käsin tai opettaa järjestelmälle erityisen sääntöjen opettajan (rule trainer) avulla. (Ingriswang ym. 2001, s. 4)

Periaatteessa Ingriswangin ja Liun (2001) AAID-järjestelmässä on samankaltaiset pääosat kuin Lanen (1998) mallissa tai Chungin ym. (1999) DEMIDS-järjestelmässä, mutta jäljitystiedon keräämisen ja poikkeamien tunnistamisen osalta toimintaperiaate eroaa, kuten jäljempänä esitetään. Järjestelmän malli ei ole Lanen (1998) ja Chung ym. (1999) esittämässä mielessä puhtaasti poikkeamaperusteinen, sillä poikkeamien tunnistus ei perustu välttämättä aikaisempaan normaalikäyttöön, vaan voi pohjautua

myös ennalta käsin määriteltyihin tunnisteiden kaltaisiin sääntöihin. Varsinaisia profiileja ei luoda, vaan sääntökannassa olevat säännöt korvaavat ne.



KUVIO 7. Relaatiotietokannan tunkeutujantunnistamisjärjestelmän yleisrakenne Ingriswangin ym. (2001, s. 4) mukaan. Välittäjä tallentaa jäljitystiedon sovellusten toiminnasta ja liikennöi tietokantaan päin. Tapahtumageneraattori muodostaa lokit välittäjän tiedoista. Lokeja verrataan tunkeutumismonitorissa sääntökantaa vastaan. Sääntökanta on muodostettu käsin tai erityistä sääntöjen opettajaa käyttämällä.

Periaatteellisella tasolla Chungin ym. (1999) järjestelmän rakenne vastaa aiemmin esitettyjen poikkeamaperusteisten järjestelmien yleistä rakennetta, sillä siinä ovat havaittavissa osat jäljitystietojen keräämistä, profiilien muodostamista ja poikkeamien tunnistamista varten. Ingriswangin ym. (2001) malli ei ole yhtä helposti tunnistettavissa poikkeamaperusteiseksi, sillä siitä puuttuvat varsinaiset profiileja luovat osat. Tutkimus kuitenkin täydentää Chungin ym. (1999) esittämiä ajatuksia tunnistamisen osalta ja esittää jäljitystietojen keräämiseen erilaisia ratkaisuja. Seuraavassa kohdassa on käsitelty tarkemmin jäljitystietojen keräämistä relaatiotietokannanhallintajärjestelmästä.

4.2 Jäljitystiedon kerääminen relaatiotietokannanhallintajärjestelmästä

Sekä Ingriswang ja Liu (2001) että Chung ym. (1999) ovat käyttäneet relaatiotietokannan väärinkäytöntunnistamisjärjestelmän rakentamisen alustana Oracle 8 –tietokannanhallintajärjestelmää, mutta jäljitystiedon keräämiseen on käytetty hyvin erilaista tapaa, kuten edellä esitetyssä arkkitehtuuriratkaisujen vertailussa on kuvattu. Eri ratkaisuille on löydettävissä perusteltuja näkökulmia ja seuraavassa tarkastelemme lähemmin em. malleissa käytettyjä jäljitystiedon keräämisen tapoja.

Chung ym. (1999) ovat käyttäneet tietokannanhallintajärjestelmän omia jäljitystoimintoja tiedon keräämiseen. Oracle–tietokannanhallintajärjestelmä tarjoaa tietyllä konfiguraatiolla mahdollisuuden jäljittää esimerkiksi käyttäjien istuntoja, kirjautumisia sekä viitattuja objekteja, kuten relaatiotietokannan tauluja (Oracle 2001b, s. 726). Mikäli kuitenkin halutaan jäljittää yksityiskohtaisemmin, kuten esimerkiksi viitattuja relaatiokannan taulun eri attribuutteja, tulee käyttää tauluille erikseen määriteltäviä herättimiä (triggers) (Oracle 2001a, s. 588). Chung ym. (1999, s. 5) eivät ole rajanneet malliaan koskemaan mitään tarkkuustasoa, vaan malli soveltuu pelkkien SQL-kyselytyyppien tarkkailemisesta aina yksittäisten attribuuttien uusien ja vanhojen arvojen tarkkailuun.

Mallissa ei oleteta, että profiilien muodostamista varten kerättäisiin tietoa kaikesta mahdollisesta, vaan jäljitettävän tiedon valinta tulee tietoturvasta vastaavan henkilön tehtäväksi. Hän valitsee tarvittavat jäljitettävät toiminnot, kohteet ja tekijät. Jäljittämistä varten valittava kokoonpano riippuu siitä, millaisen väärinkäytön tunnistamisesta ollaan erityisen kiinnostuneita. Kiinnostuksen kohteina saattavat olla vain tietyt käyttäjät, taulut tai sovellukset. (Chung ym. 1999, s. 5) Käsiteltäessä jäljitystiedon keräämistä yleisesti poikkeamaperusteisten järjestelmien yhteydessä, todettiin jo aiemmin, että liian laajamittainen jäljitystiedon kerääminen ei ole käytännössä järkevää. Myös Oraclen (2001b, s. 723) ohjeistuksessa neuvotaan valitsemaan jäljitettävät kohteet huolella ja välttämään turhan jäljitystiedon keräämistä.

Chungin ym. (1999) DEMIDS-järjestelmässä jäljitystietojen kokoamisesta pitää huolen jäljittäjä, joka käyttää hyväkseen tietokannanhallintajärjestelmän jäljitysominaisuuksia ja seuraa käyttäjien tekemiä kyselyjä tietoturvasta vastaavan henkilön tekemien

määritysten mukaan (Chung ym. 1999, s. 5). Artikkelissa ei ole kuitenkaan tarkemmin kerrottu, miten määritysten tekeminen hoidetaan. Yksinkertaisimmillaan tämä tietysti tarkoittaa tietoturvahenkilön käsin asettamia herätteitä ja tietokannan oman jäljittämisen käynnistämistä. Käytännön kannalta järkevämpi ratkaisu olisi kuitenkin jokin tietokannan kaavaa lukeva apuväline, jossa jäljitettävät toiminnot, kohteet ja käyttäjät voitaisiin valita helposti.

Lisäksi on huomattava, ettei jäljitystiedon saaminen tietokannanhallintajärjestelmästä ole lainkaan niin yksinkertaista kuin Chung ym. (1999) antavat ymmärtää. Oracle-tietokannanhallintajärjestelmä osaa tuottaa omilla jäljitysmekanismeillaan jäljitystietoja monipuolisesti, mutta ei pysty jäljittämään viitattuja taulujen attribuutteja (Oracle 2001b, s. 726). Herättimillä saadaan tieto myös viitatuista attribuuteista, mutta herättimet reagoivat ainoastaan tietokantaan kirjoittaviin operaatioihin (Oracle 2001a, s. 587), joten Chungin ym. (1999, s. 11) esimerkeissä olleet SELECT-lauseiden attribuuttilistat täytyy saada kerättyä vielä jollakin muulla tavalla. Mahdollista olisi esimerkiksi erottaa ne käyttäjien antamien ja kannan tallentamien SQL-komentojen tekstimuotoisesta esityksestä (Oracle 2001e, s. 574), mutta tästä ei ole artikkelissa mitään mainintaa. Jää myös hieman epäselväksi, onko DEMIDS-järjestelmän rakentamista todella koetettu käytännössä, koska näin merkittävä yksityiskohta on jäänyt huomaamatta.

Jäljitystieto tallennetaan DEMIDS-järjestelmässä Session-tauluun, joka sisältää kunkin jäljitystiedon identifioivan tunnuksen (TID), jäljitettävän ominaisuuden nimen (Feature) ja ominaisuuden arvon (Fvalue) (Chung ym. 1999, s. 11). Ominaisuuksia voivat olla esimerkiksi kyselyn tyyppi (queryType) ja tieto taulun (R) attribuuttiin (A) viittaamisesta. Esimerkiksi notaatio R.A = 1 tarkoittaa, että taulun attribuuttiin on viitattu ja R.A = 0 tarkoittaa, ettei siihen ole viitattu kyselyssä. Edelleen R.A.newVal = ”esimerkki” tarkoittaa, että taulun R attribuutin A uusi arvo oli kyselyssä ”esimerkki”. R.A.Val tarkoittaa vastaavasti saman attribuutin vanhaa arvoa. (Chung ym. 1999, s. 6) Edellä kuvatun kaltainen jäljitystiedon osa voisi liittyä esimerkiksi SQL:n UPDATE-lauseeseen. Ratkaisu on yleinen ja mahdollistaa helposti eri määrän attribuutteja tai erilaisia ominaisuuksia sisältävien jäljitystietojen tallentamisen (TAULUKKO 3).

DEMIDS-järjestelmässä jäljitystiedon keräämiseen kuuluu olennaisena apuvälineosana tietojenkäsittelijä, joka vastaa yksittäisten jäljitystietojen kokoamisesta jäljitysistunnoiksi. Tietojenkäsittelijä ryhmittelee jäljitystiedot istunnoiksi käyttäjän, roolin tai jonkin muun perusteen mukaan. Näin voidaan muodostaa jäljitysistunnot ja näiden avulla myöhemmin profiilit joko esimerkiksi käyttäjille tai heille kuuluville rooleille. Tietojenkäsittelijä samalla muokkaa jäljitystiedon rakennetta ja tiedon tyyppejä sekä huolehtii mm. puuttuvista arvoista. (Chung ym. 1999, s. 5)

TAULUKKO 3. Esimerkki Chung ym. (1999) mukaisesta kerätyn jäljitystiedon tallennusrakenteesta. Arvo ”1” tarkoittaa, että kohteeseen on viitattu. Tarkenne ”.Val” tarkoittaa, että jäljitetty tieto on kohteen arvo.

TID	Feature	Fvalue
1	queryType	’insert’
1	user	’user1’
1	ASIAKAS.Nimi	1
1	ASIAKAS.Osoite	1
1	ASIAKAS.Puh	1
2	queryType	’select’
2	user	’user3’
2	HLO.Nimi	1
2	HLO.Osasto	1
2	HLO.Palkka.Val	2500

Edellä käsiteltiin oppivan poikkeamapohjaisen väärinkäytön tunnistamisen heikkouksia tietojen keräysvaiheessa, jolloin järjestelmä saattaa oppia myös väärinkäyttötapauksia normaalitoimintaa mallinnettaessa. Chungin ym. (1999) mallissa oletetaan, että tietoturva-asiantuntija valvoo tietojen keräämistä ja etteivät käyttäjät väärinkäytä järjestelmää tietojenkeräysvaiheessa. Lisäksi oletetaan, että jäljitystietoa on kertynyt riittävästi profiilin muodostamista varten. Jäljitystietoa kerätään järjestelmää käytettäessä siten, että normaalit päivittäiset työrutiinit sekä esimerkiksi kuukauden alkuun ja loppuun ajallisesti sijoittuvat erityiset toimenpiteet tulevat tallennetuiksi. (Chung ym. 1999, s. 5)

Liu (2001) on tutkinut tietokantoja vastaan kohdistuvien hyökkäysten eristämistä DAIS-järjestelmällä (A Real-time Data Attack Isolation System for Commercial Database Applications). Myös aiemmin mainittu AAID-järjestelmä (Ingriswang ja Liu 2001) on osa tätä laajempaa kokonaisuutta, jonka pyrkimyksenä on luoda hyökkäyksiä sietävä tietokannanhallintajärjestelmä, ItDBMS (Intrusion tolerant Database Management System).

Liun (2001) esittämässä järjestelmässä jäljitystiedot kerätään kahdella tavalla. Tietokantaan kirjoittavat operaatiot, kuten lisäykset ja päivitykset tallennetaan herättimien avulla. Lukuoperaatioita ei kuitenkaan voida tällä tavoin tallentaa, sillä lukuoperaatioille ei voi määritellä herättimiä, kuten jo edellä käsiteltiin. Tämän vuoksi tallennus hoidetaan erikseen asiakasohjelmien ja tietokannan välissä olevalla välittäjällä. SQL-komentojen tallentaja (SQL Statement Logger) erottelee komentojen tekstimuotoisesta esityksestä kaikki lukuoperaatiot erityisten mallien (Read Set Template) avulla. (Liu 2001 s. 6) Tällainen lähestymistapa on luonteva DAIS-järjestelmän vastatoimiin keskittyvän luonteen vuoksi. Epäilyttävän käyttäjän eristämiseen suunnitellussa järjestelmässä tietokannan ja asiakasohjelman välissä toimiva välittäjä on hyvä rakenneratkaisu. Epäselväksi kuitenkin jää, miksi vain lukuoperaatiot kerätään tällä tavoin tietokannan ulkopuolella ja tietoturvanäkökulmasta huomattavasti vaarallisemmat kirjoitusoperaatiot kerätään kannan sisäisillä herättimillä. Yhtä hyvin ulkoisen välittäjän avulla voitaisiin kerätä myös kirjoitusoperaatiot. Toisaalta tekstimuotoiset SQL-kyselyt saataisiin kerättyä myös tietokannan sisällä yhdistämällä tietoa metadatanäkymistä V$SESSION (Oracle 2001e, s. 548) ja V$SQLTEXT (Oracle 2001e, s. 574), ja lukuoperaatiot voitaisiin erotella muiden kyselyjen joukosta. Tällöin koko järjestelmä toteutettaisiin kannan sisällä.

Ingriswangin ja Liun (2001) AAID-tunkeutujantunnistamisjärjestelmässä tiedot kerätään välittäjän avulla sekä suoraan tietokannasta, joka viittaa samankaltaiseen rakenteeseen edellä esitetyn DAIS-järjestelmän kanssa, vaikka juurikaan tarkempaa selvitystä jäljitystiedon keräämisestä ei ole annettu. SQL-kyselyt tallennetaan kyselyluetteloon (Statement List) ja lisäksi erotellaan lukuoperaatiot lukulokiin (Read_Log) ja kirjoitusoperaatiot kirjoituslokiin (Write_Log). AAID:n kerrotaan kuitenkin käyttävän kirjoitusoperaatioiden tallentamiseen herättimiä ja lukuoperaatioi-

den tallentamiseen tietoa transaktioista. (Ingriswang ja Liu 2001, s. 4) Ilmeisesti kyseessä on Liun (2001, s. 6) järjestelmän kaltainen ratkaisu, jossa kirjoitusoperaatiot tallennetaan herättimien avulla ja lukuoperaatiot erotellaan ja tallennetaan SQL-kyselyjen tekstimuotoisesta esityksestä.

Herättimien avulla tallennettaviin kirjoitusoperaatioihin kuuluvat SQL-komennot UPDATE, INSERT ja DELETE. Myös COMMIT ja ROLLBACK operaatiot tallennetaan. Lukuoperaatioita ovat SELECT, UPDATE, INSERT ja DELETE, sillä kirjoitusoperaatiot myös lukevat kantaa. Tallennettavien jäljitystietojen rakenne on yhtenäinen eli molemmissa lokeissa muotoa: transaktiotunnus (xid), sekvenssi (seq), toiminta (operation) ja kohde (object). Transaktiotunnus identifioi yksikäsitteisesti käyttäjän suorittaman transaktion ja sekvenssi määrää kyselylauseiden ajallisen järjestyksen. Toiminta kertoo SQL-lauseen tyypin eli suoritetun operaation ja kohde osoittaa tietokannan rakenneosan, jota toiminta koskee. Kohde voi olla esimerkiksi taulu, taulun sarake tai monikon tunnus. Myös attribuutin uusi ja vanha arvo voidaan tallentaa. (Ingriswang ja Liu 2001, s. 5) Tietyn monikon tunnistaminen tietokannasta on jo hyvin tarkan tason mallintamista, mutta voi joissain tapauksissa tietysti olla tarpeellista. Koska Ingriswangin ja Liun (2001) AAID-järjestelmä on selvästi osa isompaa kokonaisuutta, jolla pyritään luomaan hyökkäyksiä sietävä tietokannanhallintajärjestelmä, saattaa tällainen tarkempi tallentaminen olla tarpeen toipumisen ja muiden vastatoimien toteuttamiseksi.

Aiemmin käsitellyssä Denningin (1987) yleisessä poikkeamaperusteisen tunnistamisen mallissa esillä olleet kolme pääasiallista jäljitettävää ominaisuutta olivat tekijä, toiminta ja kohde. Ingriswangin ja Liun (2001) mallissa kaksi jälkimmäistä on tallennettu yhdessä luku- ja kirjoituslokeihin. Tieto tekijästä on puolestaan tallennettu erikseen tietokantaistunnoista tietoja tallentavaan transaktio- ja istuntolokiin (Trans_Session_Log). Tämä loki sisältää istunnon tunnuksen, prosessin tunnuksen, tietokannan käyttäjätunnuksen, käyttöjärjestelmän käyttäjätunnuksen, transaktion tunnuksen ja muita istuntoon liittyviä tietoja. (Ingriswang ja Liu 2001, s. 5) Liittämällä transaktio- ja istuntoloki kirjoituslokiin tai lukulokiin saadaan Denningin (1987) yleisen mallin mukaiset jäljitystiedon perusosat.

Sekä Ingriswangin ja Liun (2001) että Chungin ym. (1999) malleissa jäljitystietoa voitaisiin periaatteessa kerätä myös tapahtumien ajallisista suhteista, mutta tätä ei ole ainakaan esimerkeissä painotettu. Chung ym. (1998, s. 1) ovat todenneet, että tietokannan tietoturvakäytäntöjä määriteltäessä järjestelmän tulisi muun muassa pystyä käsittelemään tapahtumien ajallisia suhteita; väärinkäytöntunnistamisjärjestelmän pitäisi kyetä käsittelemään tapahtuman aikaa suhteessa muihin tapahtumiin, kahden tapahtuman välisten aikojen suhdetta ja jonkin tapahtuman esiintymiskertoja tietyssä aikajaksossa.

Lisäksi Chungin ym. (1999, s. 1) mukaan järjestelmä tunnistaa nimenomaan käyttäjien käyttäytymistä, eikä ainoastaan normaalia toimintaa. Kuitenkaan edellä esitetyn mukaan käyttäytymisen mallintamiselle ei ole juurikaan perusteita, sillä tietoa tallennetaan vain käyttäjän normaalista toiminnasta. Jos todella pyritään tunnistamaan käyttäytymistä ja identifioimaan käyttäjiä, pitäisi jäljitystietoa kerätä myös jostakin käyttäytymistä paremmin kuvaavasta suureesta, kuten esimerkiksi käyttäjän toimien ajallisesta järjestyksestä Lanen (1998) tapaan. Ingriswangin ja Liun (2001) mallissa ajallisen tiedon tallentamisen mielekkyys on vielä kyseenalaisempaa. Malli ei ole puhtaasti poikkeamaperusteinen, ja ainakin käsin määriteltynä ajallisia suhteita tarkkailevat säännöt olisivat vaikeita toteuttaa.

Tietokannanhallintajärjestelmän yhteyteen rakennettavassa poikkeamaperusteisessa väärinkäytöntunnistamisjärjestelmässä tärkeitä tietokannan toiminnasta jäljitettäviä tietoja ovat tekijät eli käyttäjät, toiminnot eli kyselylauseiden operaatiot sekä kohteet eli esimerkiksi tietokannan taulut, attribuutit tai tallennetut proseduurit. Tarkempaa jäljittämistä haluttaessa voidaan myös joukot eli taulujen viitatut monikot tai jopa tietokantaa muuttavien kyselyjen vanhat ja uudet attribuuttien arvot ottaa mukaan jäljittämiseen. Tietokannan järjestelmänvalvojan tulisi pystyä valitsemaan jäljitettävät kiinnostavat kohteet helposti ja tarvittaessa yksityiskohtaisesti. Mikäli käyttäjien toimien lisäksi halutaan mallintaa myös heidän käyttäytymistään, on tarpeen tallentaa myös toimien ajallisia suhteita ja mahdollisesti myös muita kuvaavia lisätietoja.

Kussakin tietokannanhallintajärjestelmässä olevat erityiset ominaisuudet määräävät, millä tavoin ja miten helposti jäljitystietoa voidaan kerätä. Jäljitystiedon kerääminen

täytyy toteuttaa kussakin järjestelmässä olemassa olevilla keinoilla, ja pahimmassa tapauksessa tämä saattaa tarkoittaa, että ainakin tarkempi jäljittäminen on mahdotonta. Herättimet ovat käytettävissä suurimmassa osassa tietokannanhallintajärjestelmiä ja ainakin osittain SQL:1999-standardia tukevien versioiden markkinoille tulo monipuolistaa jäljitystietojen keräämisen mahdollisuuksia. Parhaimmillaan tietokannanhallintajärjestelmä kerää jäljitystietoja yksinkertaisesti määrittelemällä ja tuottaa hyvin jäsenneltyjä jäljitystietoja, joiden käyttö profiilien luomisessa on suoraviivaista. Vaikeimmillaan jäljitystiedon kerääminen tarkoittaa SQL-kyselyjen tekstimuotoisten esitysten läpikäymistä, sillä tämä mahdollisuus on järjestelmästä riippumatta aina olemassa, ainakin jonkin ulkoisen ohjelmaosan avulla. Jäljitystiedon tallentaminen ja käsitteleminen sujuu joka tapauksessa tehokkaimmin ja turvallisimmin tietokannanhallintajärjestelmän sisällä, joten erillisten ohjelmien rakentaminen tätä tarkoitusta varten on melko turhaa.

Eräs huomionarvoinen näkökulma on jäljitystiedon yksityiskohtaisuuden ja hienojakoisuuden tarpeellinen taso. Edellä käsitellyissä relaatiotietokannan väärinkäytöntunnistamisjärjestelmissä on oletettu, että jäljitystietoa pitää kerätä jopa tietokannan taulujen viitatuista attribuuteista, koska taulun tarkkuus ei ole järjestelmälle riittävä. On myös esitetty, että on tarpeen kerätä tietoa käyttäjien käsittelemistä tietojoukoista eli jäljittää viitatut monikot. Relaatiotietokannanhallintajärjestelmien käyttöoikeuksien valvontajärjestelmiä koskevassa tutkimuksessaan Bertino ym. (1999, s. 105) ovat ratkaisseet sekä taulun attribuuttien alijoukon että monikkojen joukon ongelman valtuuttamalla käyttäjiä vain tauluista johdettuihin näkymiin, eikä suoraan tietokannan tauluihin. Näkymillä esitettävää tietoa voidaan rajata rakenteellisin määrittelyin käsiteltävien attribuuttien ja ehtojen avulla käsiteltävien monikkojen suhteen. Vain muutamilla tietokantaa hoitavilla käyttäjillä on oikeudet käyttää tauluja ja luoda niistä näkymiä. Muut käyttäjät toimivat ainoastaan näkymien perusteella. (Bertino ym. 1999, s. 105) Näkymä on tietokannanhallintajärjestelmän näkökulmasta monessa suhteessa tietokannan taulun kanssa yhdenvertaisessa asemassa. Näin on esimerkiksi Oraclessa jäljitystietojen keräämisen suhteen (Oracle 2001g, s. 810). Jos jäljitystietoja kerätään vain viitatuista näkymistä, se voidaan tehdä Oraclen omalla jäljitysmekanismilla ilman herättimiä ja samalla vältytään lukuoperaatioiden keräämisen ongelmalta.

Tietokantasovellusta suunniteltaessa näkymät ovat merkittävässä osassa, koska koko perustaulun tietojen näyttäminen ei useinkaan ole järkevää ja toisaalta voi olla havainnollisempaa näyttää tietoa yhdistettynä useammasta perustaulusta. Toisaalta käyttäjät harvoin käyttävät tietokantasovellusta suoraan kyselylausein, vaan tietokannan päälle rakennetaan usein graafisella käyttöliittymällä varustettu asiakassovellus, tai sovelluspalvelin välittää kannan tiedot eteenpäin esimerkiksi html-sivuna. Näkymien järkevän määrittelyn avulla sekä relaatiotietokannan käyttöoikeuksien hallinta että väärinkäytöntunnistamisen rakentaminen helpottuu merkittävästi. Jos jäljitystietoja kerätään vain kokonaisista tauluista, näkymistä ja mahdollisesti ajettavista proseduureista, tehostuu sekä jäljitystietojen keruu, profiilien luominen että väärinkäytön tunnistaminen. Mikäli väärinkäytön tunnistamisessa kuitenkin halutaan tarkastella myös tietokannan taulujen yksittäisiä attribuutteja tai niiden arvoja, on käytettävä herättimiä tai muita hienojakoista jäljitystä tukevia mekanismeja.

Myös tietokannan päälle rakennetun sovelluksen toteutuksesta riippuu, miten tehokkaasti väärinkäytön tunnistaminen voidaan rakentaa. Mikäli sovellus esimerkiksi lukee tarpeettomia attribuutteja kannasta tai päivittää kantaan joka kerta kaikki näyttöön luetut attribuutit, vaikka niitä ei olisikaan muutettu, eivät jäljitystiedot pysty kuvastamaan käyttäjän todellisia toimia ilman arvotason perusteellista tutkimista. Toisaalta sovellukset voivat oikein toteutettuina helpottaa järjestelmien rakentamista, sillä ne säännönmukaistavat käyttäjien toimintaa ja edesauttavat yhdenmukaisten ja säännöllisten jäljitystietojen keräämistä.

Sopivasti mitoitetun ja oleellisten asioiden tarkkailuun tähdätyn jäljitystietojen keräämisen pohjalta voidaan järjestelmälle opettaa normaalikäyttö tehokkaasti poikkeamaperusteista väärinkäytön tunnistamista varten. Opettamista varten tarvitaan jokin malli, jonka perusteella jäljitystiedoista johdetaan normaalikäytön profiilit. Seuraavassa on käsitelty tietokannanhallintajärjestelmien yhteyteen rakennettavan väärinkäytöntunnistamisjärjestelmän normaalikäytön profiilien rakentamismallia.

4.3 Normaalikäytön profiilien muodostaminen opetusvaiheessa

Poikkeamien ja väärinkäytön tunnistamista varten tarvitaan ensin tieto normaalista toiminnasta. Käyttöjärjestelmä- ja verkkotasolla normaalitoiminnan tallentaminen on tarkoittanut pääasiassa käyttäjän antamien komentojen tai järjestelmäkutsujen tallentamista normaalitoiminnan profiileiksi. Tietokannanhallintajärjestelmän ja sen sovellusten väärinkäytön tunnistamista varten tarvitaan tietokannan tietorakenteen ja sovellusten toiminnan huomioon ottavat normaalikäytön profiilit.

Tietokannan tiedon rakenteen, kuten taulujen attribuutit ja liitokset perusavainten (primary key) ja viiteavainten (foreign key) avulla, voi selvittää yleensä melko helposti tietokannan tietohakemistosta. Tämän lisäksi täytyisi normaalikäytön profiilin muodostamista varten tietysti tietää, mitä nimenomaisia tiedon osia käyttäjä tavallisesti käsittelee.

Haettaessa tai muokattaessa tietoa relaatiotietokannassa, SQL-kyselyn taulun sarakkeisiin viittaavat attribuutit liittyvät toisiinsa perus- ja viiteavainten välisten yhteyksien avulla. Chung ym. (1999, s. 6) määrittelevät käyttäjäkohtaisen käsitteen työskentelyala (working scope), joka tarkoittaa toisiinsa liittyvien attribuuttien joukkoa, johon viitataan kyselyssä. Esimerkkinä käyttäjä user1 lisää tietokannan asiakas-tauluun monikon SQL-kyselyllä (ESIMERKKI 1).

ESIMERKKI 1.

SQL-lause: insert into asiakas (nimi, osoite) values (’Matti Mallikas’, ’Mallikatu 1’);

Jäljitys: {queryType=’insert’, relation=’asiakas’, asiakas.nimi=1, asiakas.osoite=1, asiakas.puh=0, asiakas.nimi.Val=’’, asiakas.osoite.Val=’’, asiakas.osoite.Val=’’, asiakas.nimi.newVal=’Matti Mallikas’, asiakas.osoite.newVal=’Mallikatu 1’, asiakas.puh.newVal=’’}.

Työskentelyala:

WS_{user1} = {queryType=’insert’, asiakas.nimi=1, asiakas.osoite=1}.

Esimerkin SQL-lauseesta voidaan saada Chung ym. (1999, s. 6) mukaan jäljitystietoina yllä luetellut ominaisuudet. Jäljitystieto voidaan tämän jälkeen tiivistää käyttäjän USER1 työskentelyalaksi (WS).

Chung ym. (1999) käyttävät tunnistamismallinsa yhtenä perusteena viitattujen attribuuttien etäisyyttä tietokannan kaavassa. Rakenteellinen etäisyys (schema distance) kahden attribuutin välillä voidaan laskea niiden taulujen perus- ja viiteavaimin liittämiseksi tarvittavien liitosten minimimäärästä. Attribuuttiparien suhteellinen etäisyys (pairwise distance) saadaan sitten normalisoimalla saatu arvo kaikkien tietokannan taulujen välisellä suurimmalla etäisyydellä, jolloin arvo on välillä 0-1. Attribuuttijoukon rakenteellinen etäisyys on tästä joukosta muodostettujen attribuuttiparien suurin suhteellinen etäisyys. (Chung ym. 1999, s. 7)

Olkoon tietokannassa esimerkiksi taulut, joiden enimmäisetäisyys eli enintään tarvittava liitosten määrä on seitsemän. Tämä tarkoittaa, että minkä tahansa taulun tieto voidaan liittää minkä tahansa toisen taulun tietoon tekemällä enintään seitsemän liitosta. Oletetaan lisäksi, että käyttäjä antaa SQL-kyselyn, jossa viitataan attribuutteihin kolmesta taulusta. Jäljitystieto sisältää kaksi attribuuttia yhdestä taulusta, yksi etäisyydellä kaksi ja yksi etäisyydellä neljä. Attribuuttiparien etäisyydet toisistaan ovat 0, 2, 4 ja 0, 2, 4 ja 2, 2, 2 sekä 4, 4, 2. Tällöin suhteelliset parittaiset etäisyydet ovat vastaavasti 0, 2/7, 4/7 ja 0, 2/7, 4/7 ja 2/7, 2/7, 2/7 sekä 4/7, 4/7, 2/7. Koko attribuuttijoukon rakenteellinen etäisyys on parien maksimi eli $4/7 \approx 0{,}57$.

Tieto viitattujen attribuuttien rakenteellisesta läheisyydestä tietokannan kaavassa ei kuitenkaan riitä, vaan tarvitaan myös tieto attribuutteihin viittaamisen tavallisuudesta kaikkien jäljitystietojen suhteen. Kullekin erilaiselle jäljitystiedolle lasketaan, kuinka monta kertaa identtinen jäljitystieto esiintyy kussakin jäljitysistunnossa. Kunkin jäljitystiedon frekvenssi normalisoidaan sitten kaikkien jäljitysistunnon jäljitystietojen määrällä, jolloin saadaan laskettua kunkin jäljitystiedon tavallisuus (access affinity). (Chung ym. 1999, s. 7)

Attribuuttijoukon sisäinen etäisyysmitta (distance measure) saadaan laskemalla sopivasti painotettuna yhteen rakenteellinen etäisyys ja jäljitystiedon tavallisuus kaavalla:

$$\text{painotus} \times \text{rakenteellinen etäisyys} + (1 - \text{painotus}) \times (1 - \text{jäljitystiedon tavallisuus})$$

Painotus voi saada arvoja 0:sta 1:een ja sen arvon asettaa järjestelmän tietoturvasta vastaava henkilö. Suuremmalla painotuksella voidaan korostaa tietokannan rakenne-ominaisuuksien merkitystä etäisyysmitan laskennassa. (Chung ym. 1999, s. 8)

Jos edellisen esimerkin kaltaisessa jäljitysistunnossa on kaikkiaan 100 jäljitystietuetta ja näistä 11 on samanlaisia kuin esimerkin tietue, on jäljitystiedon tavallisuus tällöin 11/100 = 0,11. Jos oletetaan, että tietoturvasta vastaava henkilö on asettanut järjestelmän painotukseksi 0,5, saadaan kyseisen jäljitystiedon attribuuttien joukon etäisyysmitaksi $0{,}5 \times 4/7 + (1 - 0{,}5) \times (1 - 11/100) \approx 0{,}73$.

Jäljitystiedoista muodostetaan istunnoittain usein esiintyvien ominaisuuksien joukkoja (frequent itemsets), joihin kuuluvat ominaisuudet kustakin jäljitystiedosta. Identtiset jäljitystiedot muodostavat kuitenkin vain yhden ominaisuuksien joukon. Joukko koostuu jäljitystiedon ominaisuuksien lisäksi kunkin ominaisuustiedon tyypistä (domain), ko. jäljitystiedon kanssa identtisten jäljitystietojen määrästä (support) ja jäljitystiedon tauluihin viittaavien attribuuttien etäisyysmitasta. (Chung ym. 1999, s. 9)

Koska ei ole järkevää tallentaa useaan kertaan ominaisuusjoukkoja ja niiden alijoukkoja, Chung ym. (1999, s. 10) määrittelevät käsitteen minimaalinen usein esiintyvien ominaisuuksien joukko (minimal frequent itemset). Tällaiselle joukolle ei ole olemassa ylijoukkoa, joka sisältäisi kaikki siinä olevat ominaisuudet ja tällaista joukkoa voidaan käyttää edustamaan kaikkia suppeampia alijoukkoja (ESIMERKKI 2). (Chung ym. 1999, s. 10)

ESIMERKKI 2.
U_1 = {queryType=’insert’, asiakas.nimi=1, asiakas.osoite=1, asiakas.puh=1}
U_2 = {queryType=’insert’, asiakas.nimi=1, asiakas.osoite=1}
U_3 = {queryType=’insert’, asiakas.nimi=1}

Esimerkissä on tallennettu usein esiintyvien ominaisuuksien joukot U_1, U_2 ja U_3. U_3 on joukkojen U_2 ja U_1 alijoukko. U_2 on joukon U_3 ylijoukko ja joukon U_1 alijoukko. U_1 on joukkojen U_2 ja U_3 ylijoukko. Tällöin minimaalinen usein esiintyvien ominaisuuksien joukko on U_1, jonka ominaisuudet sisältävät kaikki joukkojen U_2 ja U_3 ominaisuudet.

Identtisten jäljitystietojen määrälle ja etäisyysmitalle voidaan asettaa raja-arvot, joiden mukaan jäljitystiedot joko otetaan käsittelyyn tai jätetään huomiotta. Tiukoilla raja-arvoilla eli korkealla identtisten jäljitystietojen lukumäärän vaatimuksella ja alhaisella etäisyysmitan arvolla saadaan opetusvaiheessa muodostettua hyvin tyypillisiä profiileja. Tuotanto- eli tunnistuskäytössä järjestelmän raja-arvoja väljennetään siten, että muodostuu enemmän ominaisuuksien joukkoja. (Chung ym. 1999, s. 10) On toisaalta hieman kyseenalaista jättää jokin opetusvaiheen jäljitystieto pois sen harvinaisuuden vuoksi. Kirjoittajat itse mainitsevat, että jäljitystietoa pitäisi kerätä myös organisaatioissa harvoin toistuvista operaatioista. Esimerkiksi kirjanpidon kuukausittaiset tai vuosittaiset toimenpiteet saattaisivat jäädä tällaisten raja-arvojen vuoksi profiilin muodostuksen ulkopuolelle ja aiheuttaa myöhemmin säännöllisesti vääriä hälytyksiä.

Kuten edellä käsiteltiin, jäljitysistunnot muodostetaan jonkin jäljitystiedon ominaisuuden perusteella; jäljitystiedot voidaan ryhmitellä istunnoiksi vaikkapa käyttäjien roolien perusteella. Samasta jäljitysistunnosta saadut usein esiintyvien ominaisuuksien joukot muodostavat profiilin. Jos jäljitystiedot on ryhmitelty käyttäjän mukaan, muodostuvat myös profiilit käyttäjille. (Chung ym. 1999, s. 10) Profiili on täydellinen, kun siihen on tallennettu kaikki jäljitysistunnosta muodostettavissa olevat minimaalisten usein esiintyvien ominaisuuksien joukot. (Chung ym. 1999, s. 11)

Profiiliin kuuluvat ominaisuuksien joukot tallennetaan relaatiomallin mukaan kahteen tauluun F ja FMaster siten, että F sisältää usein esiintyvien ominaisuuksien joukon tunnisteen, ominaisuuden ja sen arvon. Tässä taulussa samalla tunnisteella olevia monikkoja on yksi kutakin joukon ominaisuutta kohti. FMaster puolestaan sisältää vastaavan tunnisteen sekä kullekin joukolle lasketetut identtisten joukkojen määrän ja etäisyysmitan. (Chung ym. 1999, s. 11)

Profiilista karsitaan (prune) muodostamisen jälkeen raja-arvojen ulkopuolelle jäävät jäljitystiedot sekä ei-minimaaliset joukot. Ominaisuuksien joukot muodostavat sääntöjä, jotka määrittelevät käyttäjän toiminnan profiilin. Tarkempi DEMIDS-järjestelmässä jäljitystietojen profiileiksi muuntamiseen käytetty algoritmi löytyy Chungin ym. (1999, s. 12) artikkelista. Kirjoittajat pitävät tällaisen profiilinmuodostuksen selvänä etuna sitä,

ettei profiiliin jää redundantteja tai samaa tarkoittavia sääntöjä muiden sääntöjä päättelevien järjestelmien tapaan (Chung ym. 1999, s. 15).

Chungin ym. (1999) järjestelmä muodostaa profiilit jäljitystiedoista SQL-kyselyjä hyväksikäyttäen ja täysin järjestelmän sisällä. Profiilien rakentaminen perustuu yksinkertaisesti käyttäjän toimien tallentamiseen, joskin siinä otetaan huomioon myös tallennettavien toimintojen rakenteellinen etäisyys ja tavallisuus edellä kuvatulla tavalla.

Ingriswang ja Liun (2001, s. 4) AAID-mallissa ei varsinaisia käyttäjän normaalitoiminnan profiileja luoda, vaan poikkeamien tunnistaminen perustuu sääntöihin, joille on käsin tai automaattisesti opettamalla määritelty poikkeama-aste (anomaly degree). Kirjoittajat esittävät järjestelmän tunnistavan poikkeamia, mutta poikkeamaperusteinen tunnistaminen ei ainakaan tämän tutkielman määritelmien mukaan ole mahdollista ilman aiemmin tallennettua normaalikäyttöä. Tämän vuoksi Ingriswang ja Liun (2001) järjestelmää ei voi pitää puhtaasti poikkeamaperusteisena järjestelmänä, vaan siinä on ominaisuuksia myös tunnisteperusteisista järjestelmistä.

Automaattinen opettaminen tapahtuu järjestelmässä sääntöjen luomismoduulissa (rule generation module), jonka toimintaa ei kuitenkaan ole tarkemmin määritelty. Järjestelmän automaattisen opettamisen tutkimus on vielä kesken. Olemassa olevilla tiedonlouhintatekniikoilla pyritään rakentamaan sääntöjä, jotka kuvaisivat epänormaalia toimintaa. (Ingriswang ja Liu 2001, s. 13)

Tällainen tiedonlouhinnalla tapahtuva sääntöjen opettaminen tarvitsisi tosin opetustiedoikseen jäljitystietoja tehdyistä hyökkäyksistä tai vaihtoehtoisesti edellä esitetyn DEMIDS-järjestelmän kaltaisen normaalitoimintaa tallentavan järjestelmän, josta sitten muodostettaisiin järjestelmälle käänteisesti epänormaalia toimintaa kuvaavat käyttötilanteet. Tietokantojen monimutkaisten rakenteiden vuoksi tällaisen järjestelmän tehokkuutta on kuitenkin syytä epäillä. Muodostettuja sääntöjä tulisi hyvin paljon ja ne olisivat hyvin yksityiskohtaisia. Toinen merkittävä puute Ingriswangin ja Liun (2001) mallissa on esimerkkien perusteella käyttäjäkohtaisuuden puute eli epänormaalin toiminnan säännöt koskevat kaikkia käyttäjiä. Ilmeisesti jonkin säännön voisi luoda myös tunnistamaan käyttäjän, mutta tätä ei oltu huomioitu esimerkeissä.

Ingriswangin ja Liun (2001) ehdottama sääntöihin perustuva tunnistaminen voisi olla erinomainen tapa täydentää Chungin ym. (1999) DEMIDS-järjestelmän kaltaista poikkeamaperusteista väärinkäytön tunnistamisjärjestelmää yhdistelmäjärjestelmäksi. Aito poikkeamaperusteinen tunnistus voisi toimia väärinkäytön tunnistamisen pohjana ja erityiset käsin määriteltävät säännöt vartioisivat tärkeitä tietoja. Sääntöjä voitaisiin myös käyttää Ingriswangin ja Liun (2001, s. 9) esittämällä tavalla tarkkailemaan tietokannan sisällön tilanmuutoksia luomalla sovelluskohtaisia sääntöjä.

Lanen (1998) poikkeamaperusteisen tunnistamisen kannalta merkittävänä pitämää tapahtumien ajallista järjestystä ei ole käytetty hyväksi DEMIDS- tai AAID-järjestelmässä. Molempien järjestelmien jäljitystietoihin voidaan periaatteessa valita tallennettavaksi tieto tapahtuman hetkestä, jonka perusteella voidaan määrätä tapahtumien järjestys, mutta profiilien muodostamisen tai sääntöjen laukaisemisen esimerkeissä tätä mahdollisuutta ei ole käytetty hyväksi. Jäljitystiedoista johdetaan DEMIDS-järjestelmässä esimerkiksi käyttäjittäin ryhmitellyt profiilit, jotka kuvaavat normaalisti suoritettuja toimintoja. Toimintojen suorittamisen ajallisten suhteiden mukaan ottaminen toisi malliin tietoa myös käyttäjän käyttäytymisestä.

Jäljitystietojen perusteella opetusvaiheessa luodut profiilit kuvaavat tietokannan normaalitoiminnan jonkin jäljitystiedon ominaisuuden mukaan ryhmiteltyinä. Kun tunnistamisvaiheen jäljitystietoja verrataan aiemmin muodostettuihin profiileihin, voidaan päätellä, onko käyttö poikkeavaa vai normaalia. Seuraavassa kappaleessa on käsitelty normaalitoiminnan ja poikkeavan toiminnan luokittelua ja epäilyttävän käytön tunnistamista.

4.4 Poikkeamien tunnistaminen

Chungin ym. (1998, s. 2) mallin mukaan relaatiotietokannan poikkeamaperusteinen väärinkäytön tunnistaminen voidaan suorittaa tehokkaasti vertaamalla normaalikäytön profiileja uusiin jäljitystietoihin. Jos uudet jäljitystietueet eivät täsmää profiilien kanssa toiminta on epänormaalia. DEMIDS-järjestelmässä tunnistaja laskee kirjoittajien mukaan opetusvaiheessa tallennettuja normaalikäytön profiileja ja uusia jäljitystietoja

vertaamalla, miten epäilyttävää käyttäjän toiminta tietyllä pisteytyksellä on (Chung ym. 1999, s. 5).

Kirjoittajat (Chung ym. 1999) eivät ole kuitenkaan juuri lainkaan käsitelleet järjestelmän tunnistamisvaiheen toimintaa, eikä poikkeaman suuruuden laskentaa esitellä millään tavoin. Esimerkit puolestaan antavat ymmärtää, että perusajatus Chungin ym. (1999) mallin mukaisessa väärinkäytön tunnistamisessa on hyvin yksinkertainen. Järjestelmän on vain todettu vertaavaan profiileja uusiin jäljitystietoihin (Chung ym. 1999, s. 5), jonka voi katsoa tarkoittavan täydellisen vastaavuuden vaatimista. Toiminta on epäilyttävää ja hälytys annetaan, mikäli uudet jäljitystiedot eivät täysin vastaa aiemmin luotuja normaalikäytön profiileja. Jäljitystietoja ei kuitenkaan sellaisenaan verrata profiileihin, vaan verrattavana ovat profiilien kaltaiset ominaisuusjoukot (ESIMERKKI 3). (Chung ym. 1999, s. 10) Edellä esitetty järjestelmän rakennekuva (Chung ym. 1999, s. 4) kuitenkin antaa ymmärtää, että profiloijan läpi kulkevat vain profiileihin tallennettavat ominaisuusjoukot ja tunnistaja vertaa jäljitysistuntoja suoraan profiileihin. Ilmeisesti on kuitenkin tarkoitettu, että uusistakin jäljitysistunnoista muodostetaan profiilien kaltaisia ominaisuusjoukkoja ennen vertailua.

ESIMERKKI 3.
Usein esiintyvien ominaisuuksien joukot:
I_1 = {queryType=’insert’, asiakas.nimi=1, asiakas.osoite=1, asiakas.puh=1}
I_2 ={queryType=’select’, asiakas.nimi=1, asiakas.osoite=1, asiakas.puh=1}
I_3 = {queryType=’update’, asiakas.nimi=1, asiakas.osoite=1, asiakas.puh=1}

Uusi käyttäjä antama kysely:
select asiakas.nimi, asiakas.hetu, pankkitili.numero, pankkitili.saldo from asiakas, pankkitili where asiakas.tunnus = pankkitili.asiakastunnus

Verrattavien ominaisuuksien joukko:
I_v = {queryType=’select’, asiakas.nimi=1, asiakas.hetu=1, pankkitili.numero=1, pankkitili.saldo=1, asiakas.tunnus=1, pankkitili.asiakastunnus=1}

Esimerkin työntekijä on yksinkertaista työtehtävää toistava käyttäjä, joka yleensä lisää ja päivittää asiakkaiden yhteystietoja ja käyttää vain muutamaa kyselylausetta. Opetusvaiheessa ovat muodostuneet profiilin usein esiintyvien ominaisuuksien joukot I_1, I_2 ja I_3. Jos työntekijä järjestelmän ollessa tunnistuskäytössä päättääkin tarkastella asiakkaiden tilien tietoja, muodostaa verrattavien ominaisuuksien joukko I_v. Joukko I_v sisältää ominaisuuksia, joita ei löydy mistään profiilin joukoista I_1, I_2 tai I_3. Näin ollen uusi tehty kysely on epäilyttävä ja aiheuttaa hälytyksen.

Tunnistusvaiheessa käsittelyyn tulevat ominaisuusjoukot voidaan lisäksi valita profiilien muodostamisen lailla samojen kriteerien eli identtisten jäljitystietojen määrän ja etäisyysmitan avulla (Chung ym. 1999, s. 10). Tämä tietysti vaatii jäljitystietojen muuttamista ensin profiilien kaltaisiksi ominaisuusjoukoiksi. Kuten jo edellä käsiteltiin, tunnistettavaksi tulevien uusien jäljitystietojen karsimista niiden harvinaisuuden tai rakenteellisen etäisyyden perusteella voidaan pitää kyseenalaisena. Tällöinhän juuri väärinkäyttöyritykset saattavat jäädä järjestelmältä käsittelemättä. Raja-arvot saattavat olla tarpeen profiilien muodostamisessa, jotta saadaan muodostettua riittävän yhdenmukaiset tavallista toimintaa kuvaavat profiilit, mutta tunnistamisvaiheessa niiden käyttöä on vaikea perustella hyväksyttävästi. Tietokantajärjestelmässä yksikin tarkoin valittu SQL-lause saattaa aiheuttaa merkittäviä vahinkoja. Raja-arvoihin vertailtavien arvojen laskeminen on lisäksi mahdollista vain historiatietoa tutkivassa tunnistamisjärjestelmässä, jossa on tieto kaikista uuden jäljitysistunnon jäljitystiedoista. Reaaliaikaisessa järjestelmässä on tieto vain siihen mennessä jäljitetyistä tapahtumista, jolloin esimerkiksi etäisyysmitan laskemiseen käytetty identtisten jäljitystietojen määrä muuttuu, kun jäljitystietoa kertyy lisää.

Chung ym. (1999, s. 15) ovat havainnollistaneet tunnistamista kahdella suppealla esimerkillä, jotka esittävät ensin normaalitoiminnan profiiliin kuuluvan usein esiintyvien ominaisuuksien joukon ja sitten väärinkäyttöyrityksestä tallennettujen ominaisuuksien joukon. Esimerkeistä ensimmäinen (ESIMERKKI 4) on helposti tunnistettavissa DEMIDS-järjestelmän tunnistamisen perusajatuksen mukaiseksi, sillä käyttäjä kohdistaa viittauksensa eri taulujen attribuutteihin kuin yleensä.

ESIMERKKI 4.

$I_{Profiili}$ = {queryType=’insert’, relation=’transaction’}

I_{Uusi1} = {queryType=’select’, cc.CID=1, o.CID=1, c.CID=1, cc.CCID=1, cc.expDate=1’}

Esimerkin pankkitoimihenkilö käyttää järjestelmää yleensä maksujen lisäämiseen, mutta väärinkäyttöyritys kohdistuu asiakkaiden luottokorttitietojen selaamiseen. (Chung ym. 1999, s. 15) Tämä näkyy siten, että normaalitoiminnan profiilin ominaisuuksien joukosta ei löydy vastaavia ominaisuuksia kuin jäljitystiedoista.

Toinen Chung ym. (1999, s. 16) esittämistä esimerkeistä (ESIMERKKI 5) liittyy hienojakoisempaan väärinkäytön tunnistamiseen.

ESIMERKKI 5.

I_{Uusi2} = {queryType=’insert’, relation=’transaction’, transaction.creditSID.newVal=’badAccount’}

Siinä pankkitoimihenkilö ei vaihda viitattuja attribuutteja, mutta siirtää rahaa hyvin usein muilta tileiltä omalle tililleen. Kirjoittajat eivät kuitenkaan kerro, miten tällainen väärinkäyttö varsinaisesti tunnistetaan. Koska tässä esimerkissä myös tiedon sisältö eli hyvitettävän tilin numero otetaan tarkasteluun, ei pelkkien viitattujen attribuuttien tarkastelu riitä. Jotta voitaisiin tunnistaa jokin tiedon sisältöön määrällisesti liittyvä ominaisuus, pitää määritellä sääntö, jonka mukaan toiminta on joko epäilyttävää tai normaalia.

Poikkeamaperusteisen tunnistamisen periaatteilla profiilista voitaisiin esimerkin tapauksessa saada tietää montako kertaa pankkitoimihenkilö tietyssä aikajaksossa on opetusvaiheessa siirtänyt rahaa tililleen. Uusista jäljitystiedoista saadaan myös tieto siitä, miten usein hän siirtää rahaa tililleen tunnistusvaiheessa. Kirjoittajat eivät kuitenkaan selitä, miten näitä kahta määrää verrataan. Tällöin jää avoimeksi, mikä on järjestelmän periaatteiden mukaan ”hyvin usein” ja siis epäilyttävää. Määrän vertailussa voidaan käyttää esimerkiksi sääntöä, jonka mukaan omalle tilille ei saa tietyllä aikavälillä siirtää rahaa useampia kertoja kuin normaalitoiminnan profiiliin on

tallennettu. Lisäksi voitaisiin käyttää myös jotakin raja-arvoa. Toinen mahdollisuus olisi vertailla uusien jäljitystietojen keskimääräisiä tilisiirtomääriä pidemmällä aikavälillä normaalitoiminnan profiiliin verrattuna. Toisaalta voitaisiin tarkkailla myös kaikkien yhdelle tilille tehtyjen siirtojen keskiarvoa ja keskihajontaa, ja mikäli jollekin tilille siirtoja olisi enemmän, se voitaisiin katsoa epäilyttäväksi.

Erillistä sääntöä tarvitaan myös kertomaan järjestelmälle, keiden henkilöiden tilejä tarkkaillaan tällaisten tapahtumien varalta. Toisin sanoen tarvitaan sääntö, joka yhdistää tapahtumaa suorittavan pankkitoimihenkilön hyvitettävään tiliin, joka on hänen omansa, ja asettaa tällaiset tapahtumat tarkkailuun. Muussa tapauksessa järjestelmä tarkkailee kaikkia tapahtumia ja antaa hälytyksen, kun uuden tilin opetusvaiheen jälkeen avanneelle asiakkaalle yritetään ensimmäistä kertaa siirtää rahaa, eikä edeltäviä tapahtumia kyseiselle tilinumerolle ole olemassa. Chungin ym. (1999) toisessa esimerkissä esittämän hienojakoisemman väärinkäytöntunnistamisen selvästi tarvitsemat säännöt ovat hyvin lähellä Ingriswangin ja Liun (2001) esittämän AAID-järjestelmän periaatteita, jossa väärinkäytön tunnistaminen perustuu pelkästään kerättyyn jäljitystietoon ja sille käsin määriteltyihin sääntöihin.

Ingriswangin ja Liun (2001) esittämän mallin säännöt (ESIMERKKI 6) ovat perustaltaan herättimien kaltaisia ECA-sääntöjä (event-condition-action), jotka kertovat mitä tehdään tietyn tapahtuman jälkeen tiettyjen ehtojen ollessa voimassa. Ingriswangin ja Liun (2001, s. 9) järjestelmässä säännöt koostuvat neljästä osasta:

- Tapahtuma (event) määrittelee toiminnan ja toiminnan kohteen. Toiminta voi olla lisäys, poisto tai päivitys ja kohde tässä tapauksessa taulun monikko.
- Tärkeys (priority) kuvaa, mikä sääntö on tärkein usean säännön toimiessa samaan aikaan.
- Ehto (condition) määrittelee, minkä ehtojen pitää olla tosia taulun monikolle, jotta tapahtuma aiheuttaisi säännön laukaisemisen ja hälytyksen. Ehto voi sisältää useita funktioita, joiden kaikkien pitää palauttaa tosiarvo, jotta ehto olisi voimassa.
- Hälytys (alarm) on määritelty poikkeaman merkittävyys, jonka sääntöjen käsittelijä ilmoittaa ehdon ollessa voimassa.

ESIMERKKI 6.

- Tapahtuma: INSERT(Tilisiirrot)
- Tärkeys: 50
- Ehto: TiliasiakasOnPankkitoimihenkilo ja HyvityksiaYliKeskiarvonViikossa(5)
- Hälytys: 50

Esimerkin säännössä oleva funktio TiliasiakasOnPankkitoimihenkilö vertaa tapahtuman syöttävän pankkitoimihenkilön henkilötunnusta ja hyvitettävän tilin asiakkaan henkilötunnusta ja palauttaa tosiarvon, mikäli ne ovat samat. Toinen funktio HyvityksiäYliKeskiarvonViikossa vertaa kyseisen pankkitoimihenkilön opetusvaiheessa tälle tilille keskimäärin viikossa tekemiä tilisiirtoja ja tunnistusvaiheen kyseisen viikon tilisiirtojen määrää. Funktio palauttaa tosiarvon, mikäli niitä on enemmän kuin keskiarvo ja parametrinä annettu raja-arvo 5 yhteenlaskettuna. Hälytys laukaistaan, mikäli molemmat funktiot palauttavat tosiarvon. Eri osatapahtumien hälytysarvot summataan transaktion poikkeama-asteeksi, jota verrataan tietoturvasta vastaavan henkilön asettamaan epäilyttävän toiminnan raja-arvoon (Ingriswang ja Liu 2001, s. 4).

Hyvin hienojakoisella tasolla sovelluksen semantiikkaa hyväksikäyttävä ja tietotasoa jäljittävä väärinkäytöntunnistamisjärjestelmä vaatii edellä esitetyn kaltaisia käsin määriteltyjä sääntöjä. Relaatiotietokannanhallintajärjestelmän yhteyteen rakennettava väärinkäytöntunnistamisjärjestelmä on Chung ym. (1999) esittämän mallin tavalla yleisellä tasolla tietoinen tietokantasovelluksen rakenteesta, jos se tarkkailee viitattuja kohteita ja niille suoritettuja toimintoja. Jos kuitenkin halutaan seurata tarkemmin sisällöllisesti, mitä sovelluksella tehdään, tarvitaan tietoa sovellusalueesta ja käsin määriteltyjä sääntöjä, jotka kertovat, miten profiileja ja uusia jäljitystietoja pitää vertailla attribuuttien arvojen suhteen. Tietokannan turvallisuudesta vastaavan henkilön täytyy tällöin määritellä, riittääkö pelkkä vastaavanlaisen tapahtuman eli attribuutin arvon olemassaolo, verrataanko nykyisiä tapahtumamääriä historiallisiin, tarvitaanko raja-arvoja ja tarvitaanko lisäksi jonkin muun ehdon täyttyminen, jotta tapahtuma olisi epäilyttävä.

4.5 Yhteenveto

Chungin ym. (1999) sekä Ingriswangin ja Liun (2001) lähestymistavat väärinkäytön tunnistamiseen relaatiotietokannanhallintajärjestelmissä ovat periaatteiltaan hyvin erilaiset. Chungin ym. (1999) järjestelmä perustuu poikkeamaperusteiseen ja oletusarvoisesti kaiken kieltävään malliin, jonka ongelmia ovat väärät hälytykset. Lisäksi ainakaan mallia käsittelevä artikkeli ei pysty vakuuttamaan, että järjestelmä kykenisi tunnistamaan sellaista yksityiskohtaista väärinkäyttöä, joka voidaan havaita vain attribuuttien arvoja tutkimalla. Ingriswangin ja Liun (2001) malli puolestaan keskittyy juuri tällaisiin yksityiskohtiin ja on siksi altis hyväksymään ennalta säännöiksi määrittelemättömiä väärinkäyttöyrityksiä. Mallit ovat toisiaan täydentäviä ja käyttämällä relaatiotietokannan väärinkäytön tunnistamisjärjestelmän rakentamisessa pohjana poikkeamaperusteista järjestelmää ja lisäämällä sen yhteyteen erityisiä sovellustason väärinkäyttötapauksia tunnistavia sääntöjä, voitaisiin todennäköisesti rakentaa melko tyydyttävä yhdistelmäjärjestelmä.

Kumpikaan malli ei kuitenkaan ainakaan artikkelien perusteella sisällä Lanen (1998) tutkimaa käyttäjän käyttäytymisen mallintamista suoritettujen toimintojen ajallisen järjestyksen perusteella. Mikäli käyttäjän toiminnan lisäksi tunnistamisessa haluttaisiin käyttää hyväksi myös käyttäjän käyttäytymismalleja, voitaisiin se toteuttaa Lanen (1998) ehdottamalla tavalla. Yhtä hyvin käyttöjärjestelmän komentojen kuin tietokantakyselyidenkin ajallista järjestystä voitaisiin käyttää hyväksi tutkittaessa, onko käyttäjien toiminnan lisäksi myös heidän käyttäytymisensä toimintoja suoritettaessa normaalia.

Poikkeamaperusteisen väärinkäytöntunnistamisjärjestelmän rakentaminen relaatiotietokannanhallintajärjestelmän yhteyteen vaatii monipuolisen jäljitystietojen keräysjärjestelmän riittävän hienojakoisen mallin luomiseksi. Vaikka jäljitystietoa voidaan kerätä useilla eri tavoilla, tulisi löytää ratkaisu, jolla jäljitystietoa voitaisiin kerätä haluttaessa mahdollisimman monesta tapahtumasta. Profiilien luominen ja poikkeamien tunnistaminen voidaan toteuttaa helposti nykyisien tietokannanhallintajärjestelmien ohjelmointiominaisuuksilla. Profiileja luotaessa on kuitenkin tarkkaan harkittava, millä tarkkuustasolla ja minkä jäljitystiedon ominaisuuksien mukaan mallintaminen on järkevää tehdä. Vaikka tietokannan väärinkäytön tunnistamisessa on luontevaa käyttää

tietoja käyttäjän varsinaisesta toiminnasta, voi olla hyödyllistä seurata myös käyttäjän käyttäytymisen muutoksia. Jossain tapauksessa myös tunnisteperusteisen järjestelmän tapaan toimivat säännöt saattavat olla tarpeen tunnistamisjärjestelmän täydentämiseksi.

Seuraavassa luvussa on esitetty edellä käsitellyn perusteella hahmotellun yksinkertaisen prototyyppijärjestelmän periaatteet jäljitystiedon keräämisen, normaalitoiminnan profiilien muodostamisen ja väärinkäytön tunnistamisen osalta. Lisäksi kuvataan esimerkkinä prototyyppijärjestelmän valvoma tietokantasovellus ja sen kuvitteellinen toimintaympäristö.

5 PROTOTYYPPIJÄRJESTELMÄN KUVAUS

Relaatiotietokannanhallintajärjestelmien väärinkäytön tunnistamista koskevan kirjallisuuden vähyydestä johtuen toteutettiin kokonaiskuvan saamiseksi kevyt prototyyppi, jolla pystyttiin todentamaan tällaisen väärinkäytöntunnistamisjärjestelmän rakentamisen mahdollisuudet yhdessä tyypillisessä ja laajasti käytössä olevassa kaupallisessa relaatiotietokannanhallintajärjestelmässä. Prototyypin rakentamisella haettiin vastauksia aiemmin julkaistun ja tämän tutkielman käytettävissä olleen tutkimusmateriaalin jättämiin epäselviin kysymyksiin. Prototyypillä tutkittiin pääasiassa, miten jäljitystietoa voidaan tällaisessa järjestelmässä tuottaa ja tallentaa sekä miten profiilien muodostaminen ja väärinkäytön tunnistaminen voisi toimia tietokannanhallintajärjestelmän sisällä ilman ulkoisia ohjelmanosia.

5.1 Toteutusalustan valintaperusteet ja koeympäristön rakentaminen

Prototyypin toteutusalustaksi valittiin tutkimuksen aloitushetkellä uusin versio (9.0.1.1) Oracle 9i –tietokannanhallintajärjestelmästä. Oracle valittiin sen kaupallisen käytön yleisyyden ja tutkimustyöhön soveltuvuuden vuoksi. Oracle -tietokannanhallintajärjestelmä oli helposti saatavilla ja lisensoitavissa ilmaiseksi tutkimuskäyttöön. Myös järjestelmän laitteistovaatimukset olivat kohtuulliset ja sen palvelin- sekä asiakasohjelmanosat pystyttiin asentamaan tavalliseen mikrotietokoneeseen. Lisäksi aiemmat kokemukset Oraclesta ja vastaavista muista tuotteista tukivat valintaa helpon käyttöönoton, lähes standardien mukaisen toiminnan ja perusteellisen dokumentaation vuoksi (Oracle 2001e, s. 1411).

Sekä Chung ym. (1999) että Ingriswang ym. (2001) olivat käyttäneet aiemmassa tutkimuksessa järjestelmänsä alustana Oracle 8:aa, joskin aiempia versiota 8.1.6 tai 8.1.7. Tämä oli myös yksi ratkaiseva tekijä järjestelmän valinnassa. Samaa järjestelmää käytettäessä aiempien tutkimustulosten ymmärtäminen ja arviointi helpottui. Merkittävin ero aiemman tutkimuksen käyttämien versioiden ja tämän tutkimuksen käyttämän version välillä oli laajentuneissa jäljitystiedon keräysmahdollisuuksissa, joita käsitellään tarkemmin jäljempänä.

Koeympäristö asennettiin tavalliseen mikrotietokoneeseen, jonka tärkeimmät tekniset ominaisuudet ovat seuraavassa luettelossa:

- suoritin: Intel Pentium III 866 MHz
- keskusmuisti: 128 Mt
- kiintolevy: 20 Gt
- käyttöjärjestelmä: Microsoft Windows 2000 Professional

Oraclen dokumentaation mukaan kriittisin järjestelmän toimintaan vaikuttava tekninen ominaisuus on vapaa keskusmuisti (Oracle 2001d, s. 53). Vaikka koejärjestelmän keskusmuisti oli Oraclen ilmoittamien ohjeellisten järjestelmävaatimusten alarajalla, riitti se koejärjestelyn kaltaiseen yhden tietokantapalvelun ja muutaman asiakasohjelman tarpeisiin. Kuitenkin Oracle varasi käyttöönsä aina kaiken käytettävissä olevan muistin.

Koejärjestelyn pystytyksen alkuvaiheissa kokeiltiin asentaa Oraclen versiota 8.1.7 ja testattiin kannan luomista ja jäljitystietojen keruuta sillä, mutta Oracle 9i:n parantuneiden jäljitystietojen keräysominaisuuksien vuoksi päädyttiin tähän uudempaan versioon. Eri Oracle 9i:n asennusvaihtoehtoja kokeiltiin ja viimein päädyttiin asentamaan Enterprise Edition täydellisenä, jotta kaikki toiminnot olisivat käytettävissä. Tietokannanhallintajärjestelmän asennuspaketti ladattiin Oracle TechNet –verkkopalvelusta ja asennettiin mukana tulevilla apuohjelmilla. Myös itse tietokanta luotiin Oraclen omilla apuohjelmilla, joilla työ sujui kohtuullisen helposti lukuunottamatta muutamia järjestelmän jumiutumisia. Tietokanta toimii Windows 2000:ssa palveluna (service), joka voidaan asettaa käynnistymään ja sammumaan tietokoneen mukana tai käsikäyttöisesti. Koejärjestelyssä palvelu asetettiin käsikäytölle.

Tietokannan luomisen jälkeen järjestelmää konfiguroitiin siten, että jäljitystietojen kerääminen olisi mahdollista. Tietokanta ei normaalisti luotuna automaattisesti jäljitä käyttäjän toimintoja. Oraclen ohjelma-alihakemistosta /admin/pfile/ löytyvään tiedostoon init.ora täytyy lisätä rivi ”audit_trail=db”, jotta jäljitysominaisuudet ylipäänsä toimivat. Asetuksella ”db” jäljitystieto tallennetaan tietokannan taulukaavan SYS tauluun AUD$. Jos valitsimen arvoksi annetaan ”os” ja lisäksi asetetaan parametrille ”audit_file_dest” jokin tiedostojärjestelmän polku ja tiedostonimi, Oracle

tallentaa jäljitystiedon annettuun ulkoiseen tiedostoon tekstinä. Oletuksena jäljitystietoja ei muodosteta lainkaan. (Oracle 2001b, s. 732) Tietokantapalvelu on lisäksi käynnistettävä uudestaan, jotta jäljitys toimisi. Tämä voi olla kompastuskivi tuotantokäytössä olevalle kriittiselle tietokannalle, jota ei voida käynnistää uudelleen ilman toiminnan keskeytymistä.

Asiakasohjelmiksi asennettiin Oraclen tietokannanhallintaohjelmat DBA Studio käyttöoikeuksien määrittelyä ja Administration Assistant tietokannan palvelun käsikäyttöistä käynnistämistä ja sammuttamista varten. DBA Studiota pitäisi periaatteessa voida käyttää molempiin tarkoituksiin, mutta koekäytössä se jumiutui helposti tietokantaa sammutettaessa. Siksi kevyt Administration Assistant on tarkoitukseen parempi. Käyttöoikeuksien määrittelemiseen ja käyttäjien luomiseen DBA Studio on helppokäyttöinen apuväline, joka muun muassa antaa mahdollisuuden tehdä kopioita käyttäjistä ja heidän oikeuksistaan. Tässä yhteydessä on syytä huomauttaa, että Oraclen versio 9i ei salli kirjautumista tietokantaan tavallisena käyttäjänä SYS-tunnuksella. Kirjautuessa tulee käyttää muotoa ”SYS AS SYSDBA”.

Lisäksi asiakaspuolen ohjelmista asennettiin sovelluskehitykseen tarkoitetut Oraclen SQL*Plus ja Toad-Softin Toad. Näillä ohjelmilla ohjelmoitiin koejärjestelmän tarpeelliset ohjelmanosat ja asetettiin tarvittavat Oraclen sisäiset palvelut. Niitä käytettiin myös asiakasohjelmina konfiguraatio- ja testausvaiheessa. Toadissa on graafinen käyttöliittymä ja runsaasti ominaisuuksia, kun taas SQL*Plus on merkkipohjainen komentorivityökalu. Kuitenkin esimerkiksi tallennettujen proseduurien ajaminen sekä jäljitysasetusten tekeminen onnistui vain SQL*Plus:ssa. Jälleen on huomattava, että SQL*Plus:ssa ei pysty käynnistettäessä kirjautumaan lainkaan SYS-tunnuksella, vaan ensin pitää kirjautua muulla tunnuksella ja antaa sen jälkeen komentorivillä komento ”CONN SYS AS SYSDBA”. Toadissa ei pystytä kirjautumaan SYS-tunnuksella lainkaan. SYS-tunnusta ei tietysti välttämättä tarvita muuten kuin tarvittavien oikeuksien antamiseen jollekin muulle käyttäjälle. Järjestelmän jäljitystaulut ja myöhemmin kuvattavan hienojakoisen jäljityksen (fine grained auditing) taulut sijaitsevat SYS-taulutilassa, eikä niitä voi siirtää, joten niiden käyttöön pitää määritellä oikeudet (Oracle 2001b, s. 734). SYS-tunnusta ei pitäisi käyttää normaaliin työskentelyyn, sillä rajattomilla oikeuksilla voi saada paljon vahinkoa aikaan.

Edellä kuvatut muutamat Oracle-tietokannanhallintajärjestelmän konfiguraatioon ja käyttöön liittyvät seikat on syytä pitää mielessä väärinkäytöntunnistusjärjestelmää suunniteltaessa ja tietokantaa sen rakentamiseen valmisteltaessa. Seuraavassa kuvataan myöhemmissä kohdissa esimerkkinä käytettävä tietokantasovellus ja sen kuvitteellinen toimintaympäristö.

5.2 Esimerkkitietokantasovellus ja sen kuvitteellinen toimintaympäristö

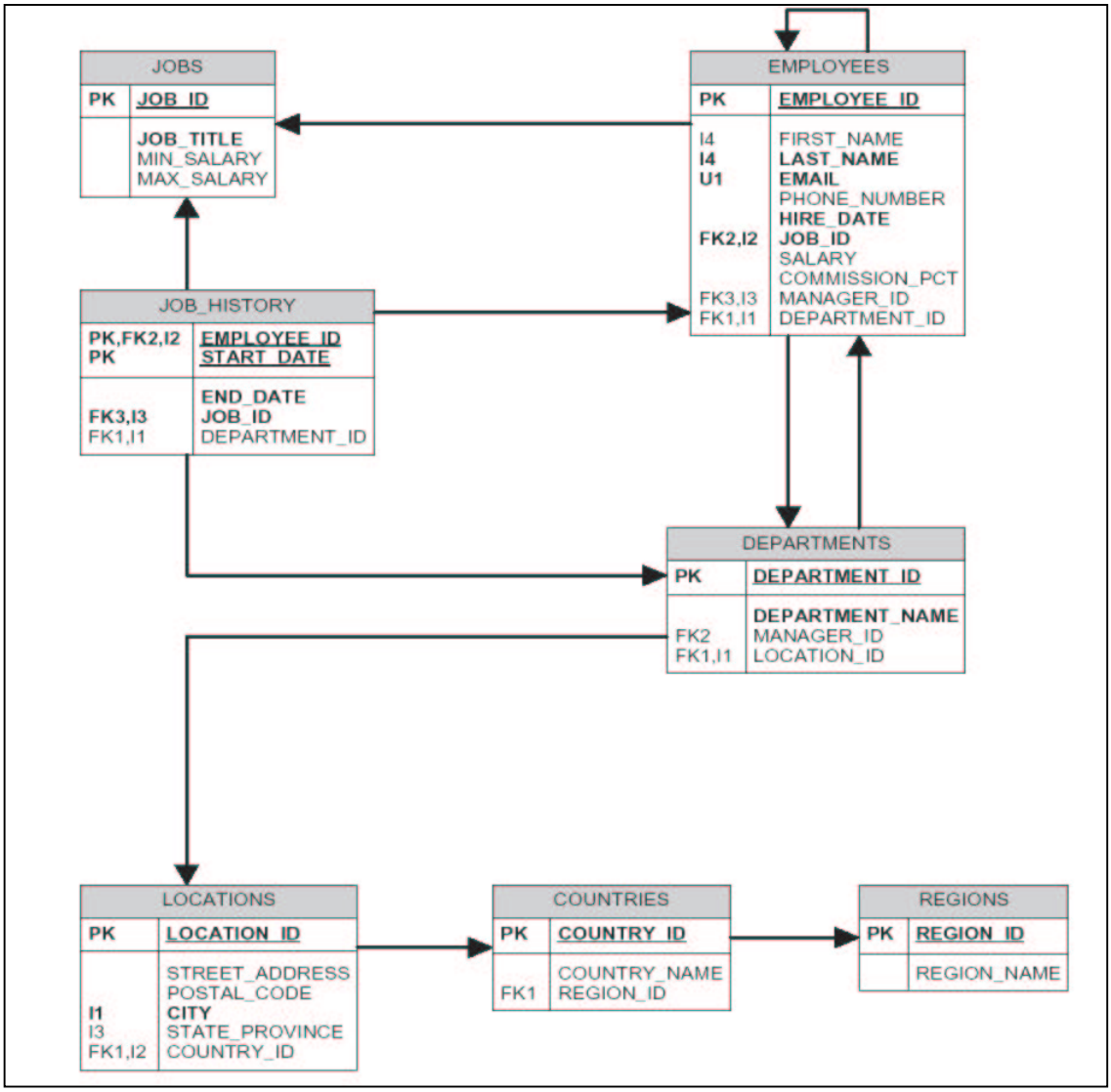
Rakennetun väärinkäytöntunnistusjärjestelmän prototyypin kokeilemista varten valittiin kaksi Oraclen esimerkkitietokantasovellusta. Järjestelmää kokeiltiin tietokantakaavoissa (schema) ”HR” (Human Resources) ja ”SH” (Sales History). Edellinen on tarkoitettu henkilöstöhallinnon tietojen tallentamista ja jälkimmäinen tuotteiden, asiakkaiden, tilausten, hintojen, kustannusten sekä mainonnan tietojen tallentamista varten. (Oracle 2001f, s. 26)

Seuraavassa on kuvattu edellä mainittuja tietokantakaavoja tarkemmin, jotta myöhempien näihin taulukaavoihin perustuvien esimerkkien ymmärtäminen olisi helpompaa. Taulukaavojen kuvioissa lihavoidut ja alleviivatut attribuutit tarkoittavat monikon yksilöiviä perusavaimia, jotka on merkitty myös tunnuksella "PK". Lihavoidut attribuutit tarkoittavat tietoja, jotka eivät voi saada tyhjäarvoa (null). Tunnuksella "U" on merkitty attribuutit, jotka voivat saada vain yksilöllisiä arvoja. Viiteavaimet on merkitty tunnuksella "FK" ja järjestysnumerolla. Indeksit on puolestaan osoitettu tunnuksella "I" ja järjestysnumerolla.

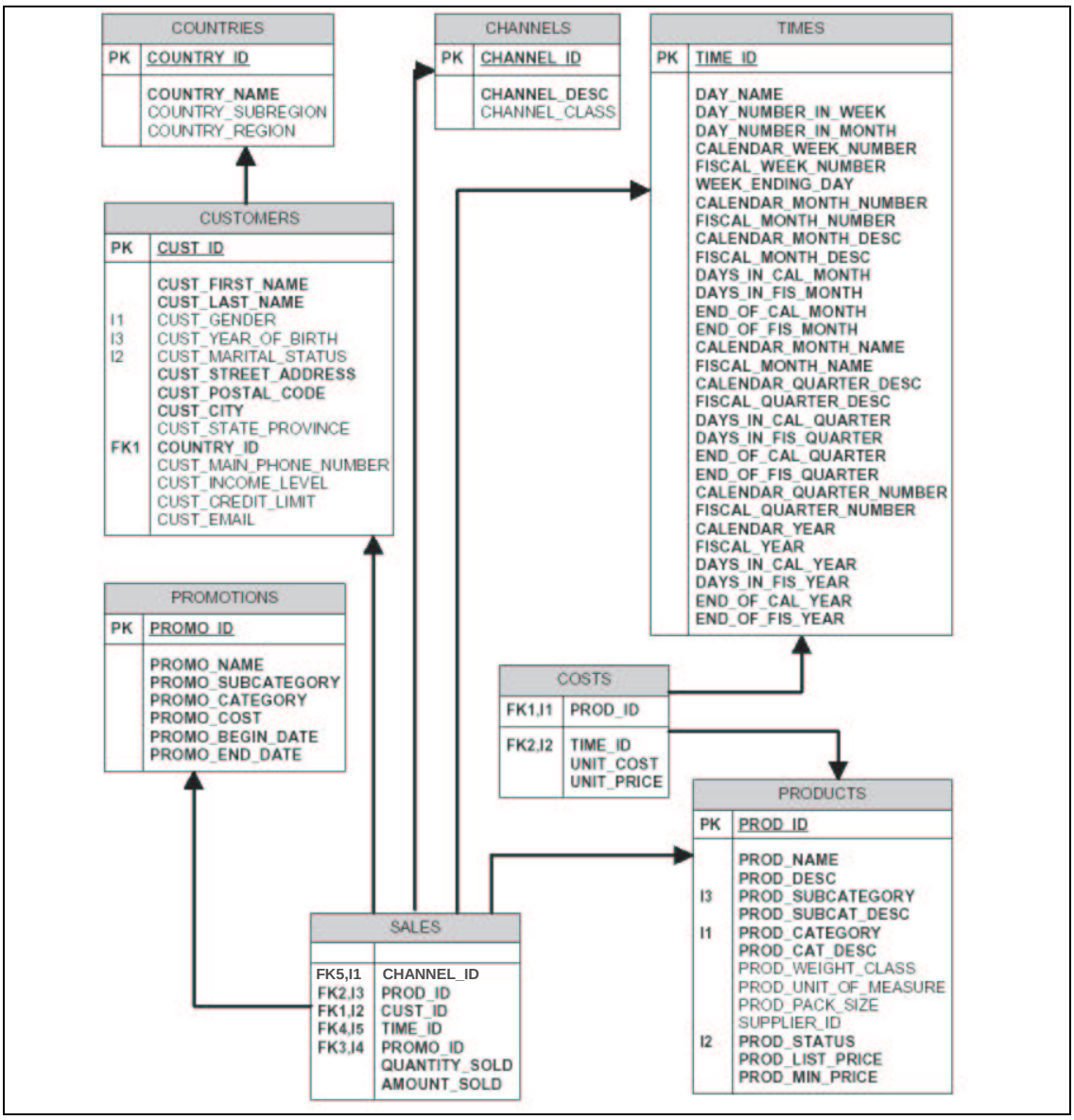
Human Resources –taulukaavan (KUVIO 8) EMPLOYEES-taulussa jokaisella työntekijällä on oma identifioiva numeronsa sekä tallennettuina nimet, yhteystiedot, palkka ja tieto esimiehestä. Jotkut työntekijät saavat palkan lisäksi provisiota, jonka laskentaperusteet tallennetaan järjestelmään. Työtehtävät sekä kunkin työtehtävän minimi- ja maksimipalkka on tallennettu JOBS-tauluun. Työntekijöiden työtehtävien vaihdokset tallennetaan tauluun JOB_HISTORY. Tiedot esimerkkiyrityksen maantieteellisestä sijoittumisesta eri kohteisiin pidetään taulussa LOCATIONS, joka puolestaan käyttää tietoja tauluista COUNTRIES ja REGIONS. Kohteesta tallennetaan katuosoite, postiosoite, kaupunki, osavaltio tai provinssi sekä maan nimi. Eri puolilla

maailmaa oleviin kohteisiin sijoittuneista osastoista pidetään kirjaa taulussa DEPARTMENTS. Jokainen työntekijä puolestaan kuuluu johonkin osastoon. (Oracle 2001f, s. 26)

Sales History –taulukaavan (KUVIO 9) tauluihin tallennetaan tietoa myynnistä ja markkinoinnista myöhempää analysointia ja päätöksentekoa varten. PRODUCTS-tauluun tallennetaan tuotteen nimi, kuvaus, kategoriat, painoluokka, mittayksikkö, pakkauskoko, toimittajan nimi, status sekä minimi- ja ohjehinnat.



KUVIO 8. Oracle 9i –tietokannan esimerkkitaulukaava ”HR”, Human Resources, esimerkkiyrityksen henkilöstöhallinnon käyttöön tallennettavaa tietoa varten (Oracle 2001f, s. 33).



KUVIO 9. Oracle 9i –tietokannan esimerkkitaulukaava ”SH”, Sales History, esimerkkiyrityksen strategisen taloushallinnon käyttöön tallennettavaa tietoa varten (Oracle 2001f, s. 37).

COSTS-tauluun tallennetaan kunkin tuotteen kustannukset määrättynä ajanhetkenä ja CUSTOMERS-taulu puolestaan sisältää asiakkaat yhteystietoineen. COUNTRIES-taulu tarkentaa asiakkaalle tallennettua maatietoa. SALES-taulu kertoo kuinka paljon, mitä tuotetta ja mihin hintaan kullekin asiakkaalle on myyty. Myyntitapahtuman yhteyteen kirjataan myös myyntiin johtanut markkinointitapahtuma, jonka tarkemmat tiedot tallennetaan tauluun PROMOTIONS. Lisäksi myynnin yhteyteen tallennetaan kunkin myyntitapahtuman kanava, joka on kuvattu taulussa CHANNELS. Raportoinnin helpottamiseksi tauluun TIMES on tallennettu aikatietoja, joiden mukaan raportit voidaan muodostaa. (Oracle 2001f, s. 29)

Esimerkkitaulukaavojen tauluja ja näkymiä on käytetty sellaisenaan, joskin taulujen ja näkymien luomista, muuntamista ja poistoa on kokeiltu jäljitystietojen keräämisen kokeilemiseksi myös kyseisten tapahtumien osalta. Prototyyppijärjestelmän kokeilussa oltaisiin voitu käyttää yhtä hyvin mitä tahansa relaatiotietokantasovellusta, mutta Oraclen valmiit esimerkkitaulut valittiin niiden helpon saatavuuden vuoksi.

Prototyyppijärjestelmän kokeiluvaihetta varten luotiin kuusi käyttäjätiliä, joille määriteltiin tyypilliset työtehtävät. Tyypillisten työtehtävien kuvausta käytettiin ohjaamaan järjestelmän käyttöä opetusvaiheessa, jotta käyttö pysyisi kullekin käyttäjälle tyypillisenä ja toistensa suhteen erilaisena. Seuraava kuvaus koejärjestelmän kuvitteellisesta toimintaympäristöstä toimi ikään kuin käytön järjestelmällisyyden varmistavana käsikirjoituksena väärinkäytöntunnistusjärjestelmän prototyypin opetusvaiheessa.

Käyttäjä Varastomies toimii yrityksen varastossa tilauksia käsittelevänä ja pakkaavana henkilönä, joka työssään hakee satunnaisesti tietoja yrityksen tietojärjestelmästä. Etupäässä hän käyttää kaavan Sales History tauluja CUSTOMERS, PRODUCTS, SALES, CHANNELS ja COUNTRIES tutkiessaan kullakin hetkellä käsittelyssä olevan tilauksen tehneen asiakkaan aiempia tilauksia ja etsiessään yhteystietoja lähteviin toimituksiin. Lisäksi Varastomies hakee silloin tällöin Human Resources –kaavan näkymästä EMP_CONTACT_VIEW tietoja yrityksen työntekijöistä, joihin joutuu ottamaan yhteyttä varmistaakseen asioita käsiteltävistä tilauksista.

Käyttäjä Palkanlaskija toimii yrityksen taloushallinto-osastolla ja vastaa työntekijöiden palkkojen maksamisesta. Työtehtäviensä hoitamiseksi Palkanlaskija hakee henkilöstöhallinnon HR-taulukaavasta työntekijöiden yhteys-, työ- ja palkkatietoja sekä päivittää silloin tällöin palkka- ja provisiotietoja. Lisäksi hän hakee provisiopalkkojen laskemiseksi myynnin tietoja SH-kaavan taulusta SALES.

Käyttäjä Myyntisihteeri lisää ja päivittää tilauksia SH-kaavan SALES-tauluun. Hän myös päivittää ja lisää tuotteiden sekä asiakkaiden tietoja tauluissa PRODUCTS ja CUSTOMERS sekä hakee tietoa tauluista CHANNELS ja COSTS avustaessaan myyntipäällikköä markkinoinnin suunnittelussa. Lisäksi Myyntisihteeri hakee silloin

tällöin muiden työntekijöiden yhteystietoja HR-kaavan näkymästä EMP_CONTACT_VIEW.

Käyttäjä Myyntipäällikkö hakee tietoja laajasti molemmista taulukaavoista tehdessään myyntityötä, suunnitellessaan markkinointia ja huoltaessaan tuotevalikoimaa. Myyntipäällikkö hakee, lisää ja päivittää tietoja SH-kaavan tauluissa PRODUCTS, PROMOTIONS, CUSTOMERS ja SALES sekä hakee tietoja tauluista CHANNELS ja COUNTRIES. Lisäksi Myyntipäällikkö hakee tietoja HR-kaavan taulusta EMPLOYEES ja näkymästä HR.EMP_CONTACT_VIEW tehdessään työnjohdollisia tehtäviä ja ottaessaan yhteyttä alaisiinsa sekä muihin yrityksen työntekijöihin.

Käyttäjä Toimitusjohtaja hakee tietoa SH-kaavan tauluista SALES, PRODUCTS, COSTS, PROMOTIONS ja COUNTRIES tehdessään yrityksen strategista suunnittelua. Lisäksi hän silloin tällöin muokkaa ja lisää tietoja tauluun PROMOTIONS suunnitellessaan markkinointia sekä päivittää tietoja taulussa COSTS tehdessään sopimuksia ostotilauksista. Lisäksi Toimitusjohtaja lisää ja muokkaa tietoja HR-kaavan tauluissa EMPLOYEES, JOBS ja JOB_HISTORY palkatessaan uusia työntekijöitä ja määrätessään työntekijöitä uusiin tehtäviin.

Käyttäjä Ylläpito käyttää järjestelmää harvoin. Tällöin hän yleensä luo, muokkaa tai poistaa näkymiä tai jopa tauluja molemmissa taulukaavoissa. Tämän lisäksi hän myös poistaa tauluista vanhoja tarpeettomia tietoja.

Prototyyppiä koestettaessa käyttäjien toimia seurattiin attribuuttitasolla, vaikka yllä olevassa kuvauksessa työtehtävät on merkitty taulu- tai näkymätasolla. Taulukossa työnkuvat ja tauluille suoritetut toimenpiteet on kuvattu vielä tiivistetysti ja täydellisenä, mutta ilman attribuutteja (TAULUKKO 4). Jos jonkin todellisen organisaation työtehtävät tietojärjestelmän suhteen olisivat näin selvästi jäsentyneet ja helposti kuvattavissa, voitaisiin periaatteessa käyttää aiemmin mainittua näkymien luontia tietyille työtehtäville. Tämän jälkeen jäljitettäisiin vain taulu- ja näkymätasolla, jolloin se voitaisiin esimerkiksi Oraclessa tehdä tietokannanhallintajärjestelmän omalla varsinaisella jäljitysmekanismilla. Koska työtehtävät ovat kuitenkin usein vaikeasti määriteltävissä, käytettiin prototyyppijärjestelmässäkin attribuuttitason jäljitystä.

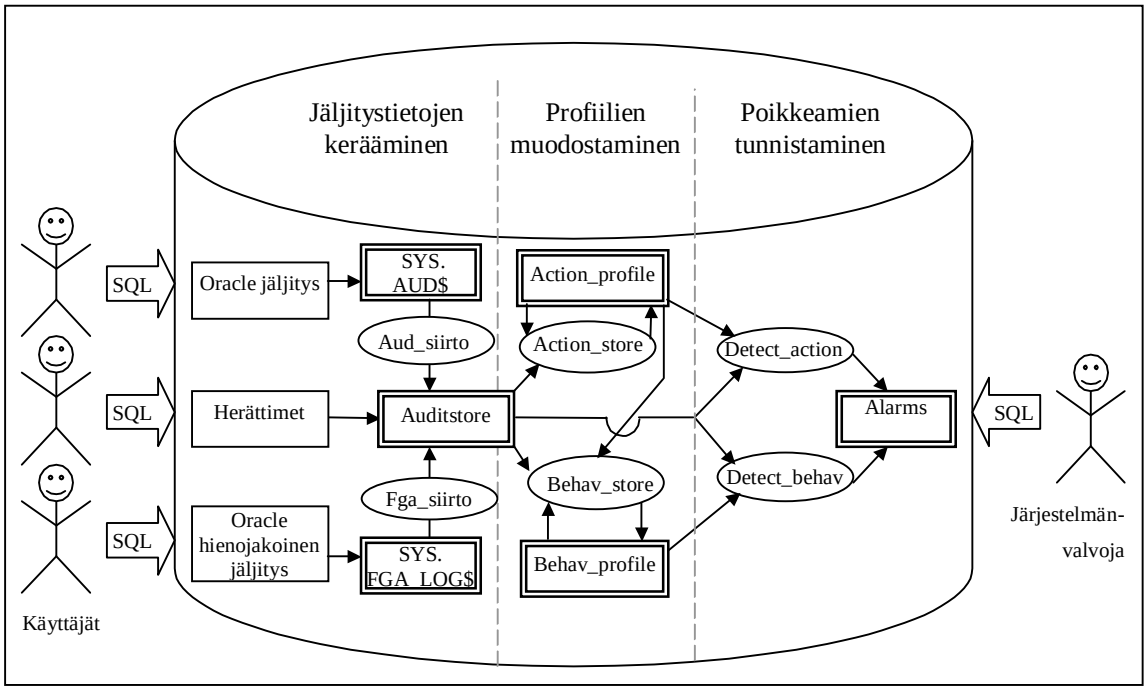
TAULUKKO 4. Prototyyppijärjestelmän kuvitteellisen toimintaympäristön käyttäjien tyypilliset toimet esimerkkitietokantasovellusta käytettäessä. Tietokannan käsittelylauseet ovat omissa sarakkeissaan, joissa kunkin käyttäjän kohdalla vastaava käytetty kohde. Rasti kohdassa DDL tarkoittaa, että käyttäjä käyttää tietokannan määrittelylauseita.

käyttäjä	select	insert	update	delete	DDL
Varastomies	sh.channels sh.customers sh.sales sh.products sh.countries hr.emp_contact_view				
Palkanlaskija	sh.sales hr.employees hr.jobs hr.job_history hr.locations		hr.employees		
Myynti-sihteeri	sh.sales sh.products sh.customers sh.channels sh.costs hr.emp_contact_view	sh.sales sh.products sh.customers	sh.sales sh.products sh.customers		
Myynti-päällikkö	sh.sales sh.products sh.customers sh.channels sh.countries sh.promotions hr.employees hr.emp_contact_view	sh.products sh.customers	sh.products sh.customers	sh.products	
Toimitus-johtaja	sh.sales sh.products sh.costs sh.countries sh.promotions	sh.promotions	sh.promotions sh.costs		
Ylläpito	sh.sales sh.times sh.costs sh.promotions			sh.sales sh.times sh.costs sh.promotions	x

5.3 Prototyyppijärjestelmän rakenne

Rakennettu prototyyppijärjestelmä jakautuu kolmeen pääosaan: jäljitystietojen keräämiseen, profiilien muodostamiseen ja poikkeamien tunnistamiseen (KUVIO 10). Kuhunkin osaan kuuluu relaatiotietokantatauluja, jotka toimivat pysyvinä tai väliaikaisina tietovarastoina sekä herättimiä ja tallennettuja proseduureja (stored procedures), jotka muodostavat, siirtävät tai muokkaavat tietoa. Lisäksi voidaan erottaa aliosia ja erillisiä ohjelmanosia ja tauluja, joilla on kokeiltu joitakin lähdeteoksissa

olleita malleja, mutta joita ei ole varsinaisesti integroitu prototyyppijärjestelmän käyttämään tunnistamismalliin.



KUVIO 10. Prototyyppijärjestelmän rakenne. Kolme pääosaa: jäljitystietojen kerääminen, profiilien muodostaminen ja poikkeamien tunnistaminen. Nuolet kuvaavat tietovirtoja, ellipsit tallennettuja proseduureja, suorakaiteet jäljitystietoa tuottavia osia ja kaksinkertaiset suorakaiteet tiedot tallentavia tietokannan tauluja.

Jäljitystietojen keräämisosa muodostuu kolmesta erillisestä alijärjestelmästä, jotka keräävät jäljitystietoa erilaisista tietokannan tietoja kyselevistä, muokkaavista ja poistavista sekä tietokannan rakennetta ja toimintaa muuntavista toimista. Oracle-tietokannanhallintajärjestelmän varsinainen jäljitysjärjestelmä on ensimmäinen näistä kolmesta alijärjestelmästä. Se tallentaa tiedot käyttäjän toimista SYS-kaavan AUD$-tauluun. Toinen alijärjestelmä on Oracle 9i-version uusi laajennus, hienojakoisen jäljityksen (fine grained auditing) ohjelmapaketti, joka pystyy jäljittämään myös SELECT-lauseissa viitatut attribuutit ilman kaikkien SQL-kyselyjen tekstimuotoisen esityksen läpikäyntiä. Se tallentaa tiedot SELECT-lauseista SYS-kaavan tauluun FGA_LOG$. Kolmas alijärjestelmä on sarja herättimiä, jotka on asetettu jokaiselle jäljitettävälle taulun attribuutille ja erikseen INSERT-, UPDATE- ja DELETE-lauseille. Ne tallentavat tiedon suoraan jäljitystietojen varsinaiseen varastoon eli SYSTEM-kaavassa sijaitsevaan AUDITSTORE-tauluun. AUD$-taulun jäljitystiedot siirretään

AUDITSTORE-tauluun käyttämällä SYSTEM-kaavaan ohjelmoitua tallennettua proseduuria AUD_SIIRTO. FGA_LOG$-taulun tiedot siirretään puolestaan tallennetulla proseduurilla FGA_SIIRTO. Molemmat proseduurit on ajastettu ajettaviksi tietyin intervallein. Jäljitystietojen keräämistä on kuvattu tarkemmin myöhemmin tässä luvussa.

Profiilien muodostamisosassa sijaitsevat taulut ACTION_PROFILE ja BEHAV_PROFILE. Edelliseen tallennetaan kunkin käyttäjän objekteille suorittamat toiminnot attribuuttitasolla ilman redundanssia. Jälkimmäiseen tallennetaan vakiomittaisina sarjoina tieto siitä, missä järjestyksessä edellä mainitut toiminnot yleensä suoritetaan. Tällä pyritään kuvaamaan käyttäjän käyttäytymistä Lanen (1998) mallissa esitetyllä tavalla. Myös tästä profiilista redundantit ajallisesti järjestetyt sarjat jätetään pois. Toiminnot tallennetaan proseduurilla ACTION_STORE, joka käy läpi AUDITSTORE-taulua ja tallentaa uudet, aiemmin profiilista puuttuvat toiminnot tauluun ACTION_PROFILE. Käyttäytyminen tallennetaan puolestaan proseduurilla BEHAV_STORE, joka lukee AUDITSTORE-taulua ja muodostaa vakiomittaisia sarjoja tunnistettuaan ACTION_PROFILE-tauluun tallennetun toiminnon. Proseduurit ACTION_STORE ja BEHAV_STORE on ajastettu ajettaviksi tietyin intervallein opetusvaiheen aikana. Tällöin profiileja rakennetaan kaiken aikaa taustalla, samalla kun käyttäjät tekevät työtä järjestelmällä. Profiilit voidaan muodostaa myös kerralla eräajona tai profilointiajo voidaan ajoittaa esimerkiksi päivittäin työajan ulkopuolelle.

Poikkeamien tunnistamisosassa proseduurit DETECT_ACTION ja DETECT_BEHAV lukevat AUDITSTORE-taulua ja vertaavat sen tietoja tauluihin ACTION_PROFILE ja BEHAV_PROFILE tallennettuihin tietoihin. Mikäli vastaavaa toimintoa tai käyttäytymistä kuvaavaa toimintojen järjestettyä sarjaa ei löydy, proseduurit tallentavat uuden jäljitystiedon tauluun ALARMS. Siitä johdetuista näkymistä yrityksen tietoturvasta vastaava henkilö voi tarkistaa poikkeamien vakavuuden ja ryhtyä tarvittaviin toimiin.

Prototyyppijärjestelmä on rakennettu täysin Oracle 9i –tietokannanhallintajärjestelmän sisälle ja jäljitystietojen keräämisessä, profiilien muodostamisessa sekä poikkeamien tunnistamisessa on pyritty mahdollisuuksien mukaan käyttämään järjestelmästä toiseen siirrettäviä ominaisuuksia. Edellä kuvattujen järjestelmän osien toimintaa on käsitelty

tarkemmin seuraavissa kolmessa kohdassa. Järjestelmän taulujen, proseduurien ja herättimien tarkoitus ja toiminta on käsitelty pääpiirteissään. Myös ne Oraclen ominaisuudet, jotka vaikuttavat tällaisen järjestelmän rakentamisratkaisuihin, on kuvattu.

5.4 Jäljitystietojen kerääminen

Kuten prototyyppijärjestelmän rakenteen kuvauksessa käsiteltiin, jäljitystietojen kerääminen tapahtuu kolmella eri tavalla: Oraclen vakiojäljitysominaisuuksilla, hienojakoisella jäljityksellä ja tietokantaan asetetuilla herättimillä. Tämän jälkeen kaikkien osajärjestelmien tulokset kerätään keskitetysti AUDITSTORE-tauluun. Prototyyppijärjestelmässä kerättävät tiedot on kiinnitetty ja jäljitystietojen varastointiin käytetyn taulun sarakkeet vastaavat kerättäviä ominaisuuksia. Järjestelmää rakennettaessa kokeiltiin myös Chungin ym. (1999) tutkimuksessa esitetyn kaltaista dynaamista jäljitystietovarastoa, jossa tallennettiin vain kolme saraketta: tapahtuman identifioiva tunnus, kerättävän tiedon eli ominaisuuden nimi ja ominaisuuden arvo. Tällaisen jäljitystiedon vertailu ja läpikäynti on kuitenkin hitaampaa ja kerättävät tiedot täytyy kuitenkin kiinnittää proseduurien ohjelmakoodeihin. Kerättävät tiedot voitaisiin tietenkin hakea jostakin ohjaustaulusta, jolloin saataisiin kohtuullisen dynaaminen järjestelmä. Jäljitystietovaraston rakenteelliset tai tehokkuuskysymykset eivät kuitenkaan olleet tutkimuksen mielenkiinnon kohteena, ja siksi tyydyttiin ratkaisuun, jossa taulun attribuutit vastaavat kerättyjä tietoja ja kerättävät tiedot on ohjelmoitu suoraan proseduureihin. Jäljitystietoja varastoivan AUDITSTORE-taulun rakenne attribuuttien selityksineen on esitetty oheisessa taulukossa (TAULUKKO 5).

Jäljitystiedosta tallennetaan aikaleima taulun sarakkeeseen TIMESTAMP, joka osoittaa jäljitystiedon muodostaneen tapahtuman hetken päivämääränä ja kellonaikana. Aikaleimaa voidaan käyttää käyttäjän toiminnan temporaalisten suhteiden mallintamiseen. Prototyyppijärjestelmän tapahtumien järjestys on kuitenkin lisäksi mallinnettu hienojakoisemmin, koska samalla ajanhetkellä saattaa muodostua usein monta jäljitystapahtumaa. Jokainen jäljitystapahtuma saa sekvenssistä AUD_SEQ oman identifioivan järjestysnumeronsa, joka tallennetaan sarakkeeseen AUD_ORDER.

TAULUKKO 5. AUDITSTORE-taulun attribuutit, niiden tyypit ja selitteet.

attribuutti	attribuutin tyyppi	selite
TIMESTAMP	DATE	jäljitystapahtuman päivämäärä ja kellonaika
USERNAME	VARCHAR2(100)	toiminnan käynnistänyt tekijä
ACTIONNO	NUMBER	toiminta tallennettuna sitä vastaavalla numerolla
SESSIONNO	NUMBER	tekijän istunnon identifioiva tunnus
ENTRYNO	NUMBER	Oraclen jäljityksen antama identifioiva numero
OBJECTNAME	VARCHAR2(100)	toiminnan kohteen nimitunnus (taulu, näkymä jne.)
SCHEMANAME	VARCHAR2(100)	toiminnan kohteen taulukaavan nimitunnus
COLUMNNAME	VARCHAR2(100)	toiminnan kohteen hienojakoisemman osan nimitunnus (taulun attribuutin nimi)
DONE	NUMBER	toiminnan tallentamisen käsiteltylippu
DONEB	NUMBER	käyttäytymisen tallentamisen käsiteltylippu
AUD_ORDER	NUMBER	jäljitystiedon identifioiva järjestysnumero

Toiminta tallennetaan sitä vastaavalla numerotunnuksella, jonka vastine löytyy Oraclen järjestelmätaulusta AUDIT_ACTIONS (LIITE 1). Esimerkiksi tunnus 1 tarkoittaa taulun SQL-lausetta CREATE TABLE ja tunnus 2 vastaavasti lausetta INSERT. Sarakkeeseen ENTRYNO tallennetaan Oraclen vakiojäljityksen antama järjestysnumero tai hienojakoisen jäljityksen antama järjestysnumero, jolla voidaan tunnistaa, onko kyseinen jäljitystieto jo siirretty alkuperäisestä taulustaan AUDITSTORE-tauluun. Vakiojäljityksessä tämän tunnistamiseksi tarvitaan lisäksi tieto käyttäjän istunnosta, joka tallennetaan numerona sarakkeeseen SESSIONNO.

Toiminnan kohde tallennetaan jäljitystiedosta riippuen yhdellä, kahdella tai kolmella attribuutilla. Ne attribuutit, jotka eivät saa arvoa korvataan merkillä ”-”. Kenttiä ei voi jättää tyhjiksi, koska Oracle ei voi vertailla tyhjiä kenttiä ja kahden tyhjän kentän vertailu palauttaa epätosiarvon. Kenttien vertailua tarvitaan esimerkiksi sen tutkimiseen, onko kyseisenlainen toiminta jo tallennettu käyttäjän profiiliin. SCHEMANAME-attribuutti tallennetaan aina, jos toiminta kohdistuu johonkin tietokannan kohteeseen, sillä se vastaa sitä taulukaavaa, jossa toiminta tapahtui. OBJECTNAME-attribuutti puolestaan tallennetaan niistä tietokannanhallintajärjestelmän toiminnoista, joilla on jokin varsinainen kohde, kuten taulu, näkymä tai proseduuri. Esimerkiksi CREATE TABLE tai DROP TABLE-lauseista ei tallenneta kuin taulukaava, kun taas INSERT ja

DELETE-lauseista tallennetaan myös taulun nimi, johon lisäys tai poisto kohdistui. SELECT- ja UPDATE-lauseista tallennetaan lisäksi myös kohdetaulun viitatut attribuutit AUDITSTORE-taulun sarakkeeseen COLUMNNAME.

Lisäksi AUDITSTORE-taulussa on kaksi lippua, jotka kuvaavat jäljitystietojen käsittelyn vaihetta prototyyppijärjestelmässä. DONE-lippu asetetaan, kun järjestelmä on tallentanut uuden toiminnan käyttäjän profiiliin tai todennut vastaavan toiminnan jo tallennetun. DONEB-lippu puolestaan asetetaan, kun käyttäjän käyttäytyminen on tallennettu sen jäljitystiedon osalta ajallisesti järjestettyinä sarjoina. Liput toimivat vastaavasti toiminnan ja käyttäytymisen käsittelyn tunnuksina myös tunnistusvaiheessa.

Oraclen vakiojäljitystoiminnot soveltuvat jäljittämään sellaisia tapahtumia, joista ei tarvitse tallentaa viitattuja attribuutteja. Tällaisia ovat esimerkiksi taulujen tai näkymien luominen, muuttaminen tai niiden poistaminen. Kuten edellä käsiteltiin, prototyypissä ei ole myöskään tallennettu viitattujen attribuuttien tietoja INSERT- tai DELETE-lauseista, mutta niiden tallentamiseen on kuitenkin käytetty herättimiä, joita käsitellään myöhemmin. Periaatteessa nämäkin toiminnot oltaisiin voitu tallentaa Oraclen vakiojäljitystä käyttäen.

Vakiojäljitystoimintojen käyttö vaatii prototyyppijärjestelmän koeympäristön kuvauksessa mainitut asetukset tietokannanhallintajärjestelmän tietoihin. Varsinaisesti jäljitys käynnistetään AUDIT-käskyllä, jota voidaan käyttää lause-, käyttöoikeus- tai kohdetasolla. Jäljitys voidaan vastaavasti lopettaa NOAUDIT-käskyllä ja käynnistystä vastaavilla tasojen lisämääreillä. Tiedot käyttäjän istunnosta, kuten kirjautumisajasta, työasemasta ja käyttöjärjestelmän käyttäjätunnuksesta voidaan jäljittää käskyllä AUDIT SESSION. (Oracle 2001b, s. 726)

Lausetasolla jäljitys kerää tietoa määritetyistä käyttäjien antamista SQL-komennoista. Tietoa kerätään kaikkien käyttäjien toimista kaikkien tietokannan kohteiden suhteen, joille kyseinen toiminta on määritelty. Esimerkiksi jäljityskäskyllä ”AUDIT TABLE” järjestelmä jäljittää käyttäjän antamia tauluihin liittyviä käskyjä, kuten ”CREATE TABLE”, ”TRUNCATE TABLE” tai ”DELETE FROM TABLE”. Käyttöoikeustasolla kerätään jäljitystietoa tiettyjen käyttöoikeuksien mahdollistamista toiminnoista kaikkien

käyttäjien ja tietokannan kohteiden suhteen. Käyttäjien joukkoa voidaan rajata käyttämällä BY-määrettä ja listaamalla käyttäjät, joiden toimista tietoa halutaan kerätä. Esimerkiksi käsky ”AUDIT CREATE ANY TRIGGER BY USER1, USER2” kerää jäljitystietoa käyttöoikeustasolla vain käyttäjien USER1 ja USER2 antamista herättimiä luovista SQL-komennoista. Kohdetason jäljitys tallentaa käyttäjien toimet aina jonkin tietokannan kohteen suhteen. Esimerkiksi käskyllä ”AUDIT DELETE ON EMPLOYEES” voitaisiin jäljittää kaikki EMPLOYEES-tauluun kohdistuvat poistokäskyt. Seurattavia käyttäjiä voitaisiin jälleen rajoittaa BY-määreellä. (Oracle 2001b, s. 726)

Vakiojäljitystä voidaan rajoittaa määreillä BY ACCESS ja BY SESSION. Edellinen asettaa järjestelmän kirjoittamaan jäljitysmonikon jokaisesta seurattavasta toiminnosta aina, kun se suoritetaan. Jälkimmäinen puolestaan luo monikon samantyyppisestä tapahtumasta vain kerran käyttäjän istunnon aikana. (Oracle 2001b, s. 727) Prototyyppijärjestelmän jäljityksen kaikki toiminnot asetettiin kirjoittamaan jäljitystieto jokaisesta suoritetusta toiminnosta, jotta myös profiilien temporaalisia malleja voitiin kokeilla.

Lisäksi jäljitystä voidaan haluttaessa rajata käsittämään vain onnistuneet tai virheeseen johtaneet käyttäjän antamat komennot. Lisämääreillä WHENEVER SUCCESSFUL tai sen negaatiolla WHENEVER NOT SUCCESSFUL voidaan tallentaa jäljitystietoa vain joko onnistuneista tai epäonnistuneista toiminnoista. (Oracle 2001b, s. 727) Poikkeamaperusteinen tunnistus tarvitsee välttämättä tietoa onnistuneista toiminnoista. Epäonnistuneiden toimintojen ottaminen mukaan jäljitykseen on lisäksi perusteltua, koska epäonnistuneet yritykset, esimerkiksi kyseisen käyttäjän käyttöoikeuksien ylitykset, ovat hyvin todennäköisiä väärinkäyttöyrityksiä. Tällaiset epäonnistuneet käyttöyritykset voitaisiin siirtää suoraan hälytyksiksi. Prototyyppijärjestelmässä ei siksi rajattu kerättävää jäljitystietoa sen onnistumisen tai epäonnistumisen perusteella.

Oraclen vakiojäljitys kerää tiedot oletuksena SYS-kaavan tauluun AUD$ (TAULUKKO 6), eikä tallennuksen kohdetaulua voi muuttaa. Taulu sisältää monipuolisesti tietoa jäljitystapahtumasta ja oheisessa taulukossa on kuvattu prototyyppijärjestelmän kannalta oleellisimmat jäljitystiedot.

TAULUKKO 6. Prototyyppijärjestelmän kannalta tärkeimmät SYS.AUD$-taulun attribuutit selitteineen.

attribuutti	attribuutin tyyppi	selite
SESSIONID	NUMBER	käyttäjän istuntoa vastaava numerotunnus
ENTRYID	NUMBER	identifioi jäljitystiedon yhdessä istunnon numeron kanssa
TIMESTAMP#	DATE	jäljitystiedon muodostaneen tapahtuman aika
USERID	VARCHAR2(30)	toiminnan käynnistäneen käyttäjän nimitunnus
ACTION#	NUMBER	toimintaa kuvaava numerotunnus
RETURNCODE	NUMBER	kertoo toiminnan onnistuneen (0) tai virhekoodin
OBJ$CREATOR	VARCHAR2(30)	kohteen omistajan/taulukaavan nimitunnus
OBJ$NAME	VARCHAR2(128)	kohteen nimitunnus

Jäljitystieto identifioidaan AUD$-taulussa attribuuttien SESSIONID ja ENTRYID yhdistelmällä, joista ensimmäinen vastaa käyttäjän kulloistakin istuntoa ja jälkimmäinen on järjestysnumero istunnon aikana tallennetuille jäljitystiedoille. TIMESTAMP#-attribuutti puolestaan kertoo jäljitystapahtuman tallennuksen päivämäärän ja ajankohdan, jota voidaan haluttaessa käyttää temporaalisiin tarkasteluihin. Prototyyppi-järjestelmässä vakiojäljityksen kautta tallennettujen toimintojen ajallinen järjestys määrätään aikaleiman ja ENTRYID-attribuutin antaman järjestyksen mukaan. USERID-attribuutti tallentaa toiminnan käynnistäneen käyttäjän nimitunnuksen, jolla jäljitystieto voidaan myöhemmin ohjata tallennettavaksi kyseisen käyttäjän profiiliin. Toiminta tallennetaan myös AUD$-taulussa numerotunnuksella, jota vastaava varsinainen toiminto löytyy Oraclen tietohakemiston taulusta AUDIT_ACTIONS. Attribuutti RETURNCODE puolestaan kertoo suoritettiinko toiminta onnistuneesti vai tapahtuiko sen aikana jokin virhe. Mikäli toiminta onnistui, sarakkeeseen tallennetaan arvo 0 ja mikäli se epäonnistui, tallennetaan vastaava virheen koodinumero. Toiminnan kohde identifioidaan taulukaavan ja kohteen nimitunnuksilla. Edellinen tallennetaan attribuut-tiin OBJ$CREATOR ja jälkimmäinen attribuuttiin OBJ$NAME.

Prototyyppijärjestelmässä käytettyjen jäljitystiedon ominaisuuksien lisäksi vakiojäljitys tallentaa tietoa vielä käyttäjän työasemasta, välissä olevasta palvelimesta, istunnon ajasta, käytetystä suoritinajasta, oikeuksien myöntäjästä ja monesta muusta seikasta, joita periaatteessa voitaisiin käyttää hyväksi normaalikäytön profiileja rakennettaessa.

Niitä ei kuitenkaan ole käytetty prototyyppijärjestelmässä, eikä niitä siksi ei käsitellä tässä tarkemmin.

Jäljitystietojen siirto AUDITSTORE-tauluun on hoidettu tietyin väliajoin automaattisesti ajettavalla tallennetulla proseduurilla AUD_SIIRTO. Siirtäminen herättimellä jokaisen jäljitystiedon tallentamisen jälkeen ei ole mahdollista, sillä Oracle ei salli herättimien määrittämistä taululle AUD$. Herättimiä ei voi ylipäänsä määritellä tauluille, jotka sijaitsevat SYS-kaavassa. Oraclen dokumentaatio ei mainitse asiaa, vaan se selvisi konstruktion rakentamisen aikana, kun siirtämistä herättimellä yritettiin. Oracle antaa tällaista herätintä luotaessa virheilmoituksen, eikä tallenna herättimen tietoja. Tämä ominaisuus pakotti miettimään uudenlaista tapaa jäljitystietojen siirtoon. Yksi mahdollisuus olisi ollut kirjoittaa tiedot käyttöjärjestelmän jäljitystiedostoon, mutta tämän jälkeen tiedot olisi pitänyt siirtää takaisin tietokantaan ulkoisen ohjelmanosan avulla. Lisäksi kokeiltiin myös INSTEAD OF -tyyppisen herättimen asettamista AUD$-taulun tiedoista toiseen taulukaavaan luodulle näkymälle (Oracle 2001a, s. 558). INSTEAD OF -tyyppinen herätin toimii kuitenkin vain laukaisevan tapahtuman käsitellessä suoraan kyseistä näkymää. Herätin ei laukea, jos toiminta kohdistuu tauluun, johon näkymä perustuu.

Siirtävä proseduuri ajetaan prototyyppijärjestelmässä tausta-ajona tietyllä väliajalla. Oracle-tietokannanhallintajärjestelmän mukana toimitetaan vakio-ohjelmapaketteja, joista paketti DBMS_JOB on tarkoitettu tausta-ajona ajettavien proseduurien määrittämiseen. Uusi tausta-ajo voidaan määrittää ajamalla paketin proseduuri SUBMIT ja antamalla sen parametreinä proseduurin nimi, ensimmäisen ajon päivämäärä ja aika (esim. SYSDATE), ajojen välillä odotettava aika vuorokausina (esim. 'SYSDATE+0.001') sekä numerotyyppinen sidosmuuttuja, joka palauttaa uuden tausta-ajon tunnusnumeron. Tausta-ajo voidaan poistaa ajamalla paketin proseduuri REMOVE, joka saa parametrikseen poistettavan tausta-ajon tunnusnumeron. Tausta-ajon tiheyttä voi muuttaa ajamalla proseduurin INTERVAL, joka puolestaan saa parametreikseen ajon tunnusnumeron sekä uuden ajojen välisen odotusajan vuorokausina. (Oracle 2001h, s. 292) Ajettaviksi asetettujen tausta-ajojen tietoja voi tarkastella SYS-kaavan DBA_JOBS-näkymässä (Oracle 2001e, s. 353). Odotusajan arvo 0.5 vastaa 12 tuntia, 0.0001 noin yhdeksää sekuntia ja 0.00001 noin yhtä sekuntia.

Kannattaa kuitenkin muistaa, että liian tiheä tausta-ajon suorittaminen kuormittaa järjestelmää.

Vakiojäljitystietoja siirtävä proseduuri AUD_SIIRTO (LIITE 2) tutkii käynnistyessään ensimmäisenä, onko uusia tallennettavia jäljitystietoja muodostunut. Mikäli uusia vakiojäljitystietoja on, siirtävä proseduuri hakee SQL-kyselyllä kursoriin ne taulun AUD$ monikot, joita vastaavia ei ole tallennettuna taulussa AUDITSTORE. Muussa tapauksessa proseduuri lopetetaan. Vertailukohtana siirrossa käytetään taulun AUD$ attribuutteja SESSIONID ja ENTRYID, jotka vastaavat attribuutteja SESSIONNO ja ENTRYNO taulussa AUDITSTORE. Siirrettäviä jäljitystietoja on rajattu siten, että vain ne jäljitystiedot, jotka ovat muodostuneet onnistuneesta suorituksesta siirretään tauluun AUDITSTORE. Lisäksi poimitaan vain toiminnan tunnistenumeroja 1, 12, 15, 21, 22, 24, 25, 68 ja 116 vastaavat jäljitystiedot. Edellä mainitut numerot vastaavat SQL-lauseita CREATE TABLE, DROP TABLE, ALTER TABLE, CREATE VIEW, DROP VIEW, CREATE PROCEDURE, ALTER PROCEDURE, DROP PROCEDURE ja EXECUTE PROCEDURE. Kyseiset komennot on valittu prototyyppijärjestelmän tallennettaviksi niiden kriittisen luonteen vuoksi. Koska kaikkea mahdollista toimintaa ei haluttu jäljittää, keskityttiin tietoa tallentaviin ja käsitteleviin kohteisiin. Prototyyppijärjestelmän kokeilun kannalta ei ole myöskään olennaista, mitkä tiedot valittiin käsiteltäviksi, sillä ne täytyy valita kulloisenkin toimintaympäristön mukaan.

Oraclen hienojakoinen jäljitys pystyy jäljittämään myös SELECT-lauseiden viitattuja attribuutteja, joita vakiojäljitys ei tallenna. Hienojakoisella jäljityksellä voidaan myös määritellä monipuolisesti ehtoja sille, missä tapauksessa jäljitystietoja muodostetaan. Hienojakoinen jäljitys on toteutettu tietokannanhallintajärjestelmän mukana toimitettavassa ohjelmapaketissa DBMS_FGA. (Oracle 2001b, s. 736) Hienojakoinen jäljitys asetetaan ajamalla paketin proseduuri ADD_POLICY, joka saa parametreikseen kohteen taulukaavan, kohteen nimen, halutun hienojakoisen jäljityksen säännön identifioivan nimen, jäljitysehdon, jäljitysattribuutin, käsittelevän proseduurin taulukaavan, käsittelevän proseduurin nimen sekä päällä-lipun. Esimerkiksi komennolla ”exec dbms_fga.add_policy(’HR’, ’COUNTRIES’, ’FGA_1’, ’COUNTRY_ID’, ’COUNTRY_ID LIKE %’);” asetetaan jäljitys, joka muodostaa jäljitystietueen, mikäli kuka tahansa käyttäjä käyttää SELECT-lausetta, joka viittaa kaavassa HR olevan taulun

COUNTRIES attribuuttiin COUNTRY_ID. Proseduuri vaatii välttämättä ehdon, joka voi olla esimerkiksi edellä esitettyä muotoa, jos kaikki tapahtumat halutaan tallentaa. Kuitenkin voitaisiin tallentaa esimerkiksi vain tiettyihin monikkoihin kohdistuvat haut muuttamalla ehtoa. Muita parametrejä ei ole välttämätöntä antaa. Käsittelevä proseduuri voidaan määrittää, mikäli yksinkertaisen lokiin kirjoittamisen sijaan halutaan välittää jäljitystiedot jollekin proseduurille. Asettamalla päällä-lipulle arvo FALSE, voidaan kantaan tallentaa asetuksia, jotka aktivoidaan vasta myöhemmin käyttöön. Jäljityssääntö voidaan poistaa ajamalla proseduuri DROP_POLICY. (Oracle 2001h, s. 254) Hienojakoisen jäljityksen asetuksia voi tarkastella SYS-kaavan taulusta FGA$, joka sisältää sääntöjen nimet, kohteet ja ehdot. Jäljitys tallentaa tiedot SYS-kaavan tauluun FGA_LOG$, johon tallentuu käyttäjän istunnon tunnus, jäljitystiedon muodostumisen ajankohta, käyttäjän nimi, käyttäjän tunnus käyttöjärjestelmässä, palvelimen nimi, päätteen nimi, kohteen taulukaava, kohteen nimi, jäljityssäännön nimi, jäljitystiedon identifioiva numero sekä SQL-lauseen teksti. Oheisessa taulukossa on lueteltu prototyyppijärjestelmän kannalta keskeisimmät sarakkeet selitteineen (TAULUKKO 7).

TAULUKKO 7. Prototyyppijärjestelmän kannalta tärkeimmät SYS.FGA_LOG$-taulun attribuutit selitteineen.

attribuutti	attribuutin tyyppi	selite
SESSIONID	NUMBER	käyttäjän istuntoa vastaava numerotunnus
TIMESTAMP#	DATE	jäljitystiedon muodostaneen tapahtuman aika
DBUID	VARCHAR2(30)	toiminnan käynnistäneen käyttäjän nimitunnus
OBJ$SCHEMA	VARCHAR2(30)	kohteen omistajan/taulukaavan nimitunnus
OBJ$NAME	VARCHAR2(128)	kohteen nimitunnus
POLICYNAME	VARCHAR(30)	jäljitysasetuksen nimitunnus
SCN	NUMBER	jäljitystiedon identifioiva järjestysnumero

On huomattava, että FGA_LOG$ ei tallenna suoraan viitattua attribuuttia, vaan se pitää selvittää yhdistämällä FGA_LOG$ ja asetustaulu FGA$ asetuksen nimen perusteella. Yhdistelty tieto viitatusta kohteesta tallennetaan sitten tauluun AUDITSTORE. Tähän käytetään proseduuria FGA_SIIRTO (LIITE 3), koska myös hienojakoisen jäljityksen lokitaulu sijaitsee taulukaavassa SYS, jolle tietokannanhallintajärjestelmä ei anna aset-

taa herättimiä. Siirtävä proseduuri ajetaan tausta-ajona AUD_SIIRTO-proseduurin tapaan ja se tarkistaa käynnistyessään, onko taulussa FGA_LOG$ uusia jäljitystietoja. Jäljitystiedon siirto tarkistetaan vertaamalla FGA_LOG$-taulun sarakkeita SESSIONID ja SCN taulun AUDITSTORE vastaaviin sarakkeisiin SESSIONNO ja ENTRYNO.

Jos uusia jäljitystietoja on, hienojakoisen jäljityksen tiedot ajasta, istunnosta, tunnuksesta, kohteesta ja käyttäjästä siirretään tauluun AUDITSTORE. Lisäksi toiminnon tunnukseksi asetetaan 3, joka vastaa tietohakemiston AUDIT_ACTIONS-taulussa SELECT-lausetta. Jos uusia tietoja ei ole, proseduuri lopetetaan.

Koska ei ole järkevää kirjoittaa yhä uudelleen edelläkuvattua monimutkaista komentoa, on prototyyppijärjestelmään ohjelmoitu proseduuri FGA_SS (LIITE 4), joka asettaa hienojakoisen jäljityksen, kun sille annetaan parametreinä jäljitettävä kaava, taulu ja attribuutti. Se tarkistaa aluksi, onko kyseinen jäljityssääntö jo olemassa. Jos ei ole, se asetetaan. Proseduuri asettaa jäljitysehdon, joka tallentaa tapahtumat kaikille monikoille, kuten edellä on esitetty. Jäljityssäännön nimi muodostetaan merkkijonosta ”FGA_ID_” sekä sekvenssistä FGA_SEQ saatavasta yksilöivästä tunnistenumerosta. Lisäksi jäljitysasetus tallennetaan tauluun FGA_SETTINGS, josta jo asetetut jäljitykset voidaan tarkistaa. Tauluun tallennetaan säännön nimi, taulukaava, kohde ja attribuutti. Jäljitys voidaan myös poistaa helposti käyttämällä proseduuria FGA_SR (LIITE 5), joka poistaa jäljitysasetuksen saatuaan parametreinä kaavan, taulun ja attribuutin nimet. Asettamisen helpottamiseksi edelleen on ohjelmoitu proseduuri FGA_TAB (LIITE 6), jolle annetaan parametreinä kaavan ja taulun nimi. Proseduuri lukee sen jälkeen attribuuttien nimet tietokannan tietohakemistosta ja asettaa jäljityksen automaattisesti käyttäen proseduuria FGA_SS, kuten edellä on kuvattu. Vastaavasti on ohjelmoitu myös proseduuri FGA_REM_TAB (LIITE 7), joka poistaa annetun taulun jäljityssäännöt.

DBMS_FGA-ohjelmapaketti jäljittää vain SELECT-lauseita, sillä muu hienojakoinen jäljitys voidaan hoitaa vastaavaan tapaan toimivilla SQL:1999-standardin mukaisilla herättimillä. Herättimet voidaan määrittää kirjoittamaan jäljitystieto suoraan AUDITSTORE-tauluun, kun käyttäjä kohdistaa INSERT-, UPDATE- tai DELETE-lauseen johonkin tietokannan taulun attribuuttiin. Herättimen laukeamiselle voidaan

asettaa ehto lisämäärellä WHEN ja sen laukaisun ajankohta voidaan määritellä ennen tapahtuman suoritusta määreellä BEFORE ja sen jälkeen määreellä AFTER. Määre INSTEAD OF on varattu näkymille suoritettujen toimintojen ohjaamiseksi muokkaamaan näkymän taustalla olevia tauluja. (Oracle 2001a, s. 558) Seuraavassa esimerkki prototyyppijärjestelmälle tyypillisen herättimen määrityksestä (ESIMERKKI 7).

ESIMERKKI 7.

```
CREATE OR REPLACE TRIGGER TRIG_10 AFTER UPDATE OF
COUNTRY_NAME ON HR.COUNTRIES
DECLARE     audsid     NUMBER;
            entid      NUMBER;
BEGIN
        SELECT SYS_CONTEXT('userenv','sessionid') INTO audsid FROM DUAL;
        SELECT SYS_CONTEXT('userenv','entryid') INTO entid FROM DUAL;
        INSERT INTO AUDITSTORE
        (TIMESTAMP,USERNAME, SESSIONNO, ENTRYNO, ACTIONNO,
        OBJECTNAME, SCHEMANAME, COLUMNNAME, AUD_ORDER)
        VALUES (sysdate, user, audsid, entid, 6, 'COUNTRIES', 'HR',
        'COUNTRY_NAME', AUD_SEQ.NEXTVAL);
END;
```

Esimerkin herätin TRIG_10 kirjoittaa jäljitystiedon AUDITSTORE-tauluun sen jälkeen, kun käyttäjä on päivittänyt COUNTRY_NAME-attribuutin arvon taulukaavan HR taulussa COUNTRIES. Muuttujiin audsid ja entid haetaan istunnon numero ja jäljitystiedon identifioiva numero ympäristömuuttujista sessionid ja entryid. AUDITSTORE-tauluun kirjoitetaan herättimen laukeamisen aikaleima, käyttäjän nimi, istunnon numero ja jäljitystiedon numero. Sarakkeeseen ACTIONNO kirjoitetaan toiminnan identifioiva numero 6, joka vastaa UPDATE-lausetta. Lisäksi tallennetaan kohteen taulukaavan nimi HR, taulun nimi COUNTRIES, attribuutin nimi COUNTRY_NAME ja tarkan jäljitysjärjestyksen määräävä numero sekvenssistä AUD_SEQ.

Koska ei ole järkevää kirjoittaa yhä uudelleen ylläkuvatun kaltaista monimutkaista komentoa herättimen asettamiseksi, on prototyyppijärjestelmään ohjelmoitu proseduuri TRIGGER_SS (LIITE 8). Se asettaa herättimet, kun sille annetaan parametreinä taulun kaava, nimi ja attribuutti, ja jos vastaavaa herätintä ei ole vielä olemassa. TRIGGER_SS tarkistaa vastaavanlaisen herättimen olemassaolon taulusta TRIG_SETTINGS, johon se

myös tallentaa tiedot asettamistaan herättimistä. Tauluun tallennetaan herättimen nimi, taulukaava, kohde, mahdollinen attribuutti sekä toimintaa vastaava numero. Proseduuri asettaa UPDATE-lauseelle herättimen, joka laukeaa viitattaessa annettuun taulun attribuuttiin sekä lisää taulun tarkkuudella tallentavat herättimet DELETE- ja INSERT-lauseille, koska niistä ei prototyyppijärjestelmässä tallenneta viitattuja attribuutteja.

Herättimien asettamisen helpottamiseksi on edelleen ohjelmoitu proseduuri TRIG_TAB (LIITE 9), joka käy läpi kannan tietohakemistoa ja asettaa herättimet kaikille annetun taulun attribuuteille. TRIG_TAB käyttää asettamiseen proseduuria TRIGGER_SS. Koska herättimien poistaminen on helppoa SQL-lauseella DROP TRIGGER, ei erityistä proseduuria niiden poistamiseksi tehty. Toisaalta sellainen voitaisiin ohjelmoida helposti samaan tapaan kuin FGA_REM_TAB.

Jäljitystietojen keräämiseen käytetään prototyyppijärjestelmässä kolmea eri tapaa. Oraclen vakiojäljitystoiminnoilla jäljitetään järjestelmää muokkaavia toimintoja, kuten taulujen luontia tai poistoa. Herättimillä kerätään tieto niistä kannan tietoja muokkaavista tapahtumista, joista halutaan tallentaa kohde attribuuttitasolla. Herättimet voidaan kuitenkin määritellä vain INSERT-, UPDATE- ja DELETE-lauseille. Oraclen ohjelmapaketissa DBMS_FGA tulevalla hienojakoisella jäljityksellä puolestaan kerätään attribuuttitason tiedot SELECT-lauseiden kohteista. Herättimet kirjoittavat jäljitystiedot suoraan AUDITSTORE-tauluun ja vakiojäljityksen sekä hienojakoisen jäljityksen tiedot siirretään omista lokeistaan ajastetuilla tallennetuilla proseduureilla. AUDITSTORE-tauluun siirron jälkeen järjestelmän kaikki kerätty jäljitystieto on yhdessä taulussa ja yhdenmukaisessa muodossa jatkokäsittelyä varten. Samaa jäljitystietojen keräystapaa käytetään sekä opetusvaiheessa profiileja luotaessa että tunnistamisvaiheessa poikkeamia etsittäessä. Seuraavassa kohdassa on käsitelty tarkemmin prototyyppijärjestelmän poikkeamien tunnistamista varten tarvitsemien profiilien muodostamista.

5.5 Normaalitoiminnan ja -käyttäytymisen profiilien muodostaminen

Jäljitystietojen keskitetyn keräämisen jälkeen voidaan opetusvaiheessa tallentaa normaalitoiminnan profiileja jonkin poikkeamaperusteisen tunnistuksen mallin

mukaisesti. Prototyyppijärjestelmässä profiileja muodostettiin koko ajan jäljitystietoja kerättäessä ja niiden rakentamisessa käytettiin yksinkertaista mallia, joka tallentaa käyttäjien suorittamat toiminnot vain kerran. Mikäli saman käyttäjän kyseinen toiminta kohteen suhteen on jo tallennettu, ei järjestelmä tallenna sitä uudelleen. Opetusvaiheen lopuksi lasketaan lisäksi kullekin käyttäjälle kunkin yksilöllisen jäljitystiedon esiintymisen frekvenssi koko jäljitysaineistossa. Yksinkertaisen aiemmin suoritettujen toimintojen tallentamisen lisäksi järjestelmä tallentaa profiilin käyttäjän käyttäytymisestä toimintojen ajallisesti järjestettyinä jaksoina. Perustana käytetään toimintaprofiilin tietoja sekä aikaleimoja ja järjestystunnuksia, joiden mukaan ajallinen järjestys voidaan päätellä.

Käyttäjän normaalitoiminta tallennetaan prototyyppijärjestelmässä proseduurilla ACTION_STORE (LIITE 10), joka on ajastettu ajettavaksi taustalla aiemmin kuvatulla tavalla. Toiminnan tallentaminen voitaisiin haluttaessa tehdä myös kerta-ajona opetusvaiheen päätyttyä. Proseduuri etsii AUDITSTORE taulusta ne yksilölliset jäljitystapahtumat, joita vastaavia ei ole tallennettu profiiliin. Erottamalla monikot, joiden DONE-lippua ei ole asetettu, tutkitaan vain aiemmin käsittelemättömät tapahtumat. Profiiliin tallentamattomat jäljitystiedot tunnistetaan liittämällä profiilitaulu ACTION_PROFILE (TAULUKKO 8) jäljitystauluun AUDITSTORE käyttämällä liitäntäattribuutteina käyttäjän nimeä, taulukaavaa, kohdetta, attribuuttia ja toiminnan numerotunnusta. Jäljitystiedot, joille ei löydy vastinetta profiilitaulusta lisätään siihen. Profiiliin lisättävä uusi monikko saa tunnusnumeron sekvenssistä TEACH_SEQ. Lopuksi kaikkien jäljitystietojen DONE-lippu asetetaan, jotta samoja tietoja ei käsitellä seuraavalla ajokerralla uudestaan.

Attribuutti USERNAME identifioi tietokannanhallintajärjestelmän käyttäjän, jonka toiminta profiiliin on tallennettu. SCHEMANAME osoittaa tietokannan taulukaavan, johon käyttäjän toiminta liittyy. Mikäli toimintaan liittyy jokin kohde, se tallennetaan profiilimonikon sarakkeeseen OBJECTNAME. COLUMNNAME-sarakkeeseen tallennetaan tapahtuman kohteeseen liittyvä attribuutti, jos se on mahdollista ja attribuutti halutaan tallentaa. ACTIONNO-sarakkeeseen tallennettava numero vastaa toiminnan numerotunnusta AUDIT_ACTIONS-taulussa. TIMES-sarakkeeseen tallennetaan opetusvaiheen lopuksi AFFINITY_STORE-proseduuria (LIITE 11)

käyttäen, kuinka monta kertaa kukin yksilöllinen jäljitystieto esiintyy aineistossa (ESIMERKKI 8). Määrästä voidaan päätellä, kuinka tavallista kyseinen toiminta on, ja haluttaessa joitakin harvinaisimpia voidaan karsia pois profiilista.

TAULUKKO 8. Käyttäjien toiminnan profiilit tallentavan taulun ACTION_PROFILE attribuutit, niiden tyypit ja selitteet.

attribuutti	attribuutin tyyppi	selite
USERNAME	VARCHAR2(40)	käyttäjän nimitunnus
SCHEMANAME	VARCHAR2(100)	taulukaavan nimitunnus
OBJECTNAME	VARCHAR2(100)	kohteen nimitunnus
COLUMNNAME	VARCHAR2(100)	attribuutin nimitunnus
ACTIONNO	NUMBER	toiminnan numerotunnus
PROFILENO	NUMBER	profiilitiedon identifioiva numerotunnus
TIMES	NUMBER	profiilitietoa vastaavien jäljitystietojen määrä jäljitystietojen joukossa

Prototyyppijärjestelmässä käytetyn profiilitaulun ominaisuudet on kiinnitetty, ja jokainen sarake vastaa tiettyä ominaisuutta. Haluttaessa voitaisiin luoda myös rakenteeltaan dynaaminen profiili käyttämällä Chungin ym. (1999) mallissa esitetyn kaltaista profiilitaulua, jossa tauluun tallennetaan profiilitiedon identifioiva tunnus, tallennettavan ominaisuuden nimi ja ominaisuuden arvo. Tällöin profiilin tiedot voivat sisältää eri määrän ominaisuuksia. Tällaista profiilitaulua (ACTION_PROFILE2) ja siihen tiedot tallentavaa proseduuria (ACTION_STORE2) kokeiltiin prototyypin rakentamisen aikana (LIITE 12). ACTION_PROFILE2 taulu voi tallentaa mitä tahansa jäljitystietoja profiileiksi, koska kaikki tiedot muutetaan merkkijonoiksi ja jäljitystiedoista tallennettavat ominaisuudet on määritelty proseduurissa ACTION_STORE2. Tallennettavat ominaisuudet voitaisiin tietysti lukea myös jonkinlaisesta määritystaulusta, jolloin järjestelmän ohjelmointia ei tarvitsisi muuttaa, jos profiilin rakentamisessa käytettäviä ominaisuuksia muutetaan. Koska järjestelmän konfiguroinnin helppoudella ei ollut prototyyppijärjestelmän tavoitteiden kannalta merkitystä, käytettiin kokeiluun profiilitaulua, jossa ominaisuudet on kiinnitetty, koska sen käsittely on helpompaa ja nopeampaa.

Chungin ym. (1999) tutkimuksessa on käyttäjän toiminnan profiilitietojen muodostuksessa ja karsinnassa käytetty hyväksi tietoa tietokantasovelluksen taulujen viitteistä toisiinsa, jonka on katsottu kuvaavan niiden rakenteellista etäisyyttä toisistaan. On hieman kyseenalaista, miten paljon tällaisen yksinkertaisen rakenteellisen ominaisuuden varaan voidaan perustaa profiilien rakentamista, mutta prototyyppijärjestelmässä kokeiltiin, miten eri taulujen ja attribuuttien etäisyydet toisistaan voitaisiin käytännössä selvittää.

Prototyyppijärjestelmään ohjelmoitiin tallennettu proseduuri REAL_DIST (LIITE 13), joka saa parametreikseen taulukaavan nimen sekä niiden kahden taulun nimet, joiden välinen etäisyys halutaan selvittää. Lisäksi asetetaan sidosattribuutti, johon selvitetty etäisyys tallennetaan. Proseduuri käyttää hyväkseen tietoja tietokannan tietohakemiston näkymistä DBA_CONSTRAINTS ja DBA_CONS_COLUMNS (Oracle 2001e, s. 344). Näistä näkymistä on edelleen johdettu näkymät REF_CON_VIEW (LIITE 13) ja REF_CON_VIEW2 (LIITE 13), jotka osoittavat taulukaavan ja mitkä kaksi taulua voidaan liittää toisiinsa jonkin REFERENCES-määritteen avulla suuntaan tai toiseen. Näiden yhdistelmä REF_CON_VIEW_UNION (LIITE 13) sisältää kaikki mahdolliset kahden taulun liitokset molempiin suuntiin.

Jos molempien attribuutteina annettujen taulujen nimi on sama, proseduuri palauttaa etäisyyden nolla ja se lopetetaan. Muussa tapauksessa proseduuri ottaa näistä monikoista käsittelyyn ne, jotka vastaavat parametrinä annettua taulukaavaa. Tämän jälkeen se hakee ne monikot, joiden ensimmäinen tauluattribuutti vastaa ensimmäistä parametrinä annettua taulun nimeä. Jos monikon toinen taulun nimi vastaa toista parametrinä annettua taulun nimeä, etäisyyden arvoksi tulee yksi. Jos taulun nimeä ei löydy joukosta, kaikkiin taulunimiin liitetään yhä uudelleen yhdistelmänäkymän monikot, kunnes löytyy monikko, jonka toisen taulun nimi vastaa parametrinä annettua taulun nimeä. Kierrokset lasketaan ja etäisyydeksi tulee tehtyjen liitosten määrä, joka on pienin määrä liitoksia, joilla annetut kaksi taulua voidaan liittää toisiinsa. Annetut kaavan ja taulun nimet sekä laskettu etäisyys tallennetaan tauluun DIST_RESULT, jotta seuraavan kerran samaa kahden taulun välistä etäisyyttä tarvittaessa sitä ei tarvitse laskea uudelleen, vaan proseduuri palauttaa sen suoraan.

Oletetaan esimerkkinä, että haluttaisiin selvittää prototyyppijärjestelmässä käytetyn esimerkkitaulukaavan Human Resources attribuuttien JOB_HISTORY.START_DATE ja REGION.REGION_ID rakenteellinen etäisyys. Taulukaavan diagrammista on helppo laskea, että tarvittavien liitosten pienin määrä on neljä. Kuitenkin tämän laskemiseksi tietohakemistosta pitää hakea tiedot kaikista kyseisen taulukaavan REFERENCES-määritteistä. Viiteavain FK1 taulussa JOB_HISTORY kuvaa, että se voidaan liittää tauluun DEPARTMENTS. Proseduuri REAL_DIST löytää liitettäviksi myös taulut JOBS ja EMPLOYEES. Seuraavalla kierroksella proseduuri löytää yhteyden taulujen DEPARTMENTS ja LOCATIONS välillä. Samalla se liittää taulun JOBS tauluun EMPLOYEES ja taulun EMPLOYEES tauluun DEPARTMENTS. Proseduuri jatkaa edelleen liittämistä, kunnes on ketjuttanut taulut LOCATIONS ja COUNTRIES ja lopulta taulut COUNTRIES ja REGIONS, jossa attribuutti REGION_ID sijaitsee. Attribuutit voitaisiin yhdistää myös vaihtoehtoisilla tavoilla, mutta tällöin ketjun pituudeksi tulisi viisi tai kuusi. Näitä vaihtoehtoja ei kuitenkaan käsitellä, sillä proseduuri lopettaa neljän liitoksen jälkeen, jolloin taulun JOB_HISTORY yhteys tauluun REGIONS löytyy.

Kahden taulun välistä etäisyyttä ei ole prototyyppijärjestelmässä käytetty hyväksi profiileja muodostettaessa, mutta etäisyyden laskemista kokeiltiin kuitenkin sen mahdollisuuksien selvittämiseksi. Taulujen ja niiden attribuuttien välistä etäisyyttä voitaisiin hyödyntää, jos prototyyppijärjestelmän profiilien rakentamiseen käytettäisiin Chung ym. (1999) esittämän kaltaista mallia. Samaa perusajatusta voitaisiin soveltaa myös siten, että prototyyppijärjestelmän profiilista karsittaisiin pois ne jäljitystiedot, jotka ovat rakenteellisesti etäällä jäljitystietojen pääjoukosta. Tällaista profiilien karsimista ei kuitenkaan prototyyppijärjestelmässä kokeiltu.

Käyttäjän normaalitoimintaa kuvaavan profiilin lisäksi rakennetaan profiili käyttäjän käyttäytymisestä ajallisesti järjestettyinä jaksoina. Perusteina käytetään ACTION_PROFILE-tauluun tallennettuja jäljitystietoja sekä AUDITSTORE-taulussa olevaa tietoa siitä, missä järjestyksessä käyttäjä on toimintoja suorittanut. Käyttäytymisprofiili tallennetaan tauluun BEHAV_PROFILE (TAULUKKO 9) vakiomittaisina jaksoina, kuten Lane (1998) ja Hofmeyer ym. (1998) ovat esittäneet. Jaksojen muodostamisen perusteina käytetään kuitenkin yksityiskohtaisempia ja enemmän

sovelluksen semantiikkaa sisältäviä jäljitystietoja kuin yllämainituissa malleissa, joissa tallennettiin järjestelmäkutsuja tai käyttäjien antamia komentoja sellaisinaan.

TAULUKKO 9. BEHAV_PROFILE-taulun attribuutit, niiden tyypit ja käyttötarkoitukset.

attribuutti	attribuutin tyyppi	selite
BEHAVNO	NUMBER	käyttäytymistiedon identifioiva numero
USERNAME	VARCHAR(40)	käyttäjä, jota tieto koskee
BEHAVIOUR	VARCHAR(100)	käyttäytymistä kuvaava käyttäjän toiminnan tunnuksista muodostettu merkkijono
TIMES	NUMBER	vastaavien käyttäytymisjaksojen määrä ko. käyttäjän jäljitystiedoissa

Prototyyppijärjestelmässä käyttäytymisjakson rakenneosa on ACTION_PROFILE-tauluun tallennettu tieto käyttäjän normaalitoiminnasta, jota kuvaa sille annettu identifioiva tunnusnumero. AUDITSTORE-taulua käydään läpi BEHAV_STORE-proseduurilla (LIITE 14), ja jokaiselle jäljitystiedolle etsitään sitä vastaava tallennettu tunnus käyttäjän toimintaprofiilista. Proseduuri hakee ensin käyttäjät, joille on muodostunut uusia jäljitystietoja. Tämän jälkeen tutkitaan käyttäjittäin, montako ajallisesti järjestettyä jaksoa jäljitystiedoista voidaan muodostaa. Jos uusia jäljitystietoja on riittävästi, niille etsitään toimintaprofiilista tunnusnumerot, joita tallennetaan peräkkäin välilyönnein eroteltuina jakson pituuden verran. Tämän jälkeen tallennetaan uusi jakso liu'uttaen ikkunaa jäljitystietojen joukossa, kuten Lanen (1998) mallissa. Tämä tarkoittaa, että ensimmäinen edellisen jakson tunnuksista jätetään pois ja mukaan otetaan yksi uusi tunnus. Jaksoja tallennetaan ja ikkunaa liu'utetaan kunnes kaikki jäljitystiedot on käsitelty (ESIMERKKI 8). Jäljitystieto, jota ei enää oteta mukaan uuteen jaksoon, merkitään käsitellyksi asettamalla AUDITSTORE-taulun DONEB-lippu. Uusi käyttäytymisjakso tallennetaan, mikäli se on erilainen kuin käyttäytymisprofiilissa olevat. Tällainen uusi jakso saa identifioivan tunnistenumeron sekvenssistä BEHAV_SEQ. Jos samanlainen jakso on jo tallennettu, vastaavan profiilimonikon TIMES-attribuutin arvoa lisätään yhdellä. Kun yhden käyttäjän käyttäytyminen on tallennettu, siirrytään seuraavaan ja tallennetaan jaksoja, kunnes kaikki käyttäjät ja jäljitystiedot on käsitelty.

ESIMERKKI 8.

AUDITSTORE (käyttäjän toiminnasta opetusvaiheessa tallennetut jäljitystiedot)

username	actionno	schemaname	objectname	columnname	aud_order
myyntisihteeri	3	sh	products	prod_name	188531
myyntisihteeri	3	sh	products	prod_desc	188532
myyntisihteeri	3	sh	products	prod_list_price	188534
myyntisihteeri	3	sh	products	prod_min_price	188535
myyntisihteeri	6	sh	sales	quantity_sold	189756
myyntisihteeri	6	sh	sales	amount_sold	189757
myyntisihteeri	3	sh	products	prod_name	189891

ACTION_PROFILE (jäljitystiedoista muodostettu normaalitoiminnan profiili)

profileno	username	actionno	schemaname	objectname	columnname	times
15432	myyntisihteeri	3	sh	products	prod_name	2
15433	myyntisihteeri	3	sh	products	prod_desc	1
15434	myyntisihteeri	3	sh	products	prod_list_price	1
15435	myyntisihteeri	3	sh	products	prod_min_price	1
15727	myyntisihteeri	6	sh	sales	quantity_sold	1
15728	myyntisihteeri	6	sh	sales	amount_sold	1

BEHAV_PROFILE (jäljitystietojen ajallisen järjestyksen perusteella muodostettu normaalin käyttäytymisen profiili, jakson pituus p=5)

behav_no	username	behaviour	times
1345	myyntisihteeri	15432 15433 15434 15435 15727	1
1346	myyntisihteeri	15433 15434 15435 15727 15728	1
1347	myyntisihteeri	15434 15435 15727 15728 15432	1

Esimerkissä käyttäjän MYYNTISIHTEERI toiminnasta on tallennettu AUDITSTORE-tauluun seitsemän jäljitystietomonikkoa, joista kuusi on yksilöllisiä. Kuudesta yksilöllisestä jäljitystiedosta on tämän jälkeen opetusvaiheessa muodostettu käyttäjän normaalitoiminnan profiili tauluun ACTION_PROFILE. Koska jäljitystiedoissa esiintyi samaa käyttäjää, toimintaa ja kohdetta koskevia jäljitystietoja kaksi kappaletta, on niiden määrä tallennettu profiilin sarakkeeseen TIMES. Lisäksi tauluun BEHAV_PROFILE on tallennettu käyttäjän normaalikäyttäytymisen profiili tallentamalla toimintojen suorittamisen järjestys jaksoina. Jaksoja muodostuu seitsemästä jäljitystiedosta ikkunan pituudella p = 5 kolme kappaletta.

Prototyyppijärjestelmä rakentaa jäljitystiedoista kaksiosaisen profiilin, joka kuvaa sekä käyttäjän normaalisti suorittamia toimintoja että käyttäytymistä toimintojen suorittamisjärjestyksen perusteella. Tällä tavoin käyttäjän toiminta voidaan mallintaa hyvin hienojakoisella tasolla, jolloin poikkeamat normaalitoiminnasta voidaan helposti havaita ja mahdollisen väärinkäyttöyrityksen kohde nopeasti selvittää. Lisäksi profiilin toinen osa pyrkii tallentamaan käyttäjälle tyypillisen järjestyksen suorittaa työtehtävät eli se mallintaa ajallista käyttäytymistä toiseen osaprofiiliin tallennettujen toimintojen suhteen. Tällöin samasta jäljitystiedosta voidaan johtaa sekä toimintaa että käyttäytymistä mallintavat profiilit. Käyttäytymisprofiili vie lisäksi vähän tallennuskapasiteettia, sillä se tallentaa vain viitteitä toisen profiilin tietoihin.

Prototyyppijärjestelmän profiilien muodostaminen perustuu pitkälti jäljitystietojen rakenteeseen, jolloin poikkeamien tunnistaminen on suoraviivaista ja yksinkertaista. Seuraavassa kohdassa on yksityiskohtaisemmin tarkasteltu, miten prototyyppijärjestelmä tunnistaa toiminnan ja käyttäytymisen poikkeamia vertaamalla uusia jäljitystietoja aiemmin tallennettuihin profiileihin.

5.6 Poikkeamien tunnistaminen

Prototyyppijärjestelmässä poikkeamien tunnistaminen on ohjelmoitu kahteen tallennettuun proseduuriin, joita ajetaan taustalla tietyin väliajoin. Tällöin väärinkäytön tunnistaminen ei ole aivan reaaliaikaista, mutta toisaalta se pitää tietokantasovelluksen toiminnan riippumattomana väärinkäytöntunnistamisjärjestelmästä. Herättimien avulla jokaisen käyttäjän suorittaman toiminnon jälkeen ajettava tunnistus olisi reaaliaikainen, mutta samalla se estäisi tietokantasovelluksen normaalin käytön, mikäli tunnistamisjärjestelmään tulisi jokin häiriö. Tietokannanhallintajärjestelmä peruuttaa koko transaktion eli myös herättimen laukaisseen toiminnon, mikäli herättimen suoritus epäonnistuu. Tausta-ajona järjestetty tunnistaminen on eristetty tietokantasovelluksesta ja se soveltuu järjestelmään, jossa päämääränä on käyttäjien toiminnan tarkkailu. Jos kuitenkin haluttaisiin estää epäilyttävät toimet ja käyttäytyminen, voitaisiin kaikessa tiedon käsittelyssä ja siirrossa käyttää myös herättimiä. Tällöin tunnistamisjärjestelmä voisi poikkeaman havaitessaan aiheuttaa virheen, joka estäisi myös jäljitystiedon keräämisen laukaisseen toiminnan suorittamisen. Tällainen järjestely on kuitenkin hidas ja virhealtis

ja ainakin prototyyppijärjestelmässä pidensi tietokantasovelluksen tapahtumien käsittelyaikoja havaittavasti. Tausta-ajona ajettavassa tunnistuksessa reaaliaikaisuuden ja järjestelmän kuormituksen suhdetta voidaan säätää tunnistuksen intervallia säätämällä.

Varsinainen poikkeamien tunnistaminen on prototyyppijärjestelmässä varsin suoraviivaista ja yksinkertaista. Käyttäjän toiminnan poikkeamat tunnistetaan proseduurilla DETECT_ACTION (LIITE 15), joka hakee taulusta AUDITSTORE käsittelyyn ne jäljitystiedon monikot, joiden DONE-lippua ei ole asetettu. Jäljitystietoja verrataan profiiliin SQL-kyselyllä, jolla etsitään taulusta AUDITSTORE monikkoja, joille ei löydy vastinetta taulusta ACTION_PROFILE. Liitäntäattribuutteina käytetään käyttäjän nimeä, toiminnan numerotunnusta, taulukaavan nimeä, kohteen nimeä sekä attribuutin nimeä. Jäljitysmonikot, joille ei löydy vastinetta profiilista, kirjoitetaan tauluun ALARMS. Tauluun tallennetaan tapahtuman aikaleima, käyttäjän nimi, istunnon tunnus, toiminnan numerotunnus, taulukaavan nimi, kohteen nimi sekä attribuutin nimi (ESIMERKKI 9). Tietoturvasta vastaava henkilö voi tarkastella suoraan taulua ALARMS tai jotakin siitä johdettua näkymää. Kun uudet jäljitystiedot on verrattu profiilin kanssa, ne merkitään käsitellyiksi asettamalla DONE-lippu.

Esimerkissä käyttäjän MYYNTISIHTEERI toiminnasta tallennetaan yhdeksän jäljitystietoa, joista kuusi on kyseisen käyttäjän normaalitoiminnan profiilia vastaavia. Kolmelle jäljitystiedolle ei kuitenkaan löydy vastinetta profiilista ja ne siirretään siksi tauluun ALARMS. Tietoturvasta vastaava henkilö voi taulun tietoja tarkastellessaan ryhtyä tarvittaviin toimenpiteisiin. Haluttaessa voitaisiin ohjelmoida lisäksi herätin, joka epäilyttävän toiminnan tietojen tallentuessa katkaisisi käyttäjän yhteyden tietokantaan ja lukitsisi käyttäjän tilin. Kyseiset vastatoimet voitaisiin laukaista ALARMS-taulun tietoihin perustuen ja suorittaa tietokannanhallintajärjestelmän ominaisuuksia hyödyntäen. Esimerkissä tauluista on esitetty vain oleellisimmat attribuutit. Lisäksi käyttäjästä saadaan lisätietoja esimerkiksi työasemasta, palvelimesta ja kirjautumisajoista, jos SYS.AUD$-tauluun AUDIT SESSION –käskyllä tallentuvat istunnon tiedot liitetään istunnon tunnuksen perusteella hälytystietoihin.

ESIMERKKI 9.

AUDITSTORE (käyttäjän toiminnasta tunnistamisvaiheessa tallennetut jäljitystiedot)

username	actionno	schemaname	objectname	columnname	aud_order
myyntisihteeri	3	sh	products	prod_name	190622
myyntisihteeri	3	sh	products	prod_desc	190623
myyntisihteeri	3	sh	products	prod_list_price	190624
myyntisihteeri	3	sh	products	prod_min_price	190625
myyntisihteeri	3	hr	employees	employee_id	190954
myyntisihteeri	3	hr	employees	salary	190955
myyntisihteeri	6	hr	employees	salary	190956
myyntisihteeri	6	sh	sales	quantity_sold	191902
myyntisihteeri	6	sh	sales	amount_sold	191903

ACTION_PROFILE (opetusvaiheen jäljitystiedoista muodostettu normaalitoiminnan profiili)

profileno	username	actionno	schemaname	objectname	columnname	times
15432	myyntisihteeri	3	sh	products	prod_name	2
15433	myyntisihteeri	3	sh	products	prod_desc	1
15434	myyntisihteeri	3	sh	products	prod_list_price	1
15435	myyntisihteeri	3	sh	products	prod_min_price	1
15727	myyntisihteeri	6	sh	sales	quantity_sold	1
15728	myyntisihteeri	6	sh	sales	amount_sold	1

ALARMS (jäljitystietomonikot, joille ei löydy vastinetta profiilista)

timestamp	username	session	actionno	schemaname	objectname	columnname
2.7.2002 15:21:33	myyntisihteeri	213	3	hr	employees	employee_id
2.7.2002 15:21:33	myyntisihteeri	213	3	hr	employees	salary
2.7.2002 15:22:54	myyntisihteeri	213	6	hr	employees	salary

Haluttaessa näytettävien käyttäjän toiminnan poikkeamatietojen määrää voitaisiin rajoittaa esimerkiksi käyttämällä aiemmin esiteltyä proseduuria REAL_DIST, joka laskee kahden taulun etäisyyden REFERENCES-määritteiden perusteella. Havaittujen poikkeamien etäisyys lähimmästä profiilitiedosta voitaisiin laskea ja näyttää poikkeama vain, mikäli se on kauempana profiilista kuin ennalta asetettu raja-arvo. Periaate on kuitenkin hieman kyseenalainen, sillä kuten Chungin ym. (1999) mallissakin, ei ole

kiistatonta, miten hyvin yksinkertainen rakenteellinen etäisyys voi kuvata jonkin tapahtuman merkittävyyttä.

Käyttäjän käyttäytymistä valvoo toinen tallennettu proseduuri, DETECT_BEHAV (LIITE 16), joka muodostaa AUDITSTORE-taulun tiedoista käyttäytymisprofiilin kaltaisia uusia jaksoja ja vertaa niitä profiiliin. Proseduuri ottaa käsittelyyn ne uudet jäljitystiedot, joiden DONEB-lippua ei ole asetettu. Tämän jälkeen proseduuri hakee kursoriin kaikki ne yksilölliset käyttäjänimet, jotka esiintyvät uusien jäljitystietojen joukossa. Tämän jälkeen jäljitystietojen joukosta tutkitaan käyttäjä kerrallaan, onko uusia jäljitystietoja riittävä määrä käyttäytymisjakson muodostamiseen. Prototyyppijärjestelmän tapauksessa tarvittava määrä on jakson pituus p = 5. Samalla lasketaan, montako jaksoa jäljitystiedoista voidaan muodostaa eli montako jaksoa pitää luoda, jotta kaikki uudet jäljitystiedot tulevat käsitellyiksi. Uudet jaksot luodaan muodostamalla merkkijono käyttäjän normaalitoiminnan profiilin tunnusnumeroista välilyönnein erotettuina. Proseduuri etsii uutta jäljitystietoa vastaavan tunnusnumeron ja tallentaa sen jakson merkkijonoon. Kun jaksossa on sen vakiopituutta vastaava määrä toimintoja, sitä verrataan käyttäytymisprofiilin tauluun kyseiselle käyttäjälle tallennettuihin jaksoihin. Mikäli vastaava jakso löytyy, käyttäytyminen on normaalia, muussa tapauksessa se on epäilyttävää ja tällöin uusi jakso siirretään ALARMS-tauluun tarkempaa tutkimista varten (ESIMERKKI 10).

ESIMERKKI 10.

AUDITSTORE (käyttäjän toiminnasta tunnistamisvaiheessa tallennetut jäljitystiedot)

username	actionno	schemaname	objectname	columnname	aud_order
myyntisihteeri	3	sh	products	prod_name	190622
myyntisihteeri	3	sh	products	prod_desc	190623
myyntisihteeri	3	sh	products	prod_list_price	190624
myyntisihteeri	3	sh	products	prod_min_price	190625
myyntisihteeri	6	sh	sales	quantity_sold	191902
myyntisihteeri	6	sh	sales	amount_sold	191903
myyntisihteeri	6	sh	sales	quantity_sold	191915
myyntisihteeri	6	sh	sales	amount_sold	191916

BEHAV_PROFILE (käyttäjän käyttäytymisestä opetusvaiheessa profiiliin tallennetut jaksot)

behav_no	username	behaviour	times
1345	myyntisihteeri	15432 15433 15434 15435 15727	1
1346	myyntisihteeri	15433 15434 15435 15727 15728	1
1347	myyntisihteeri	15434 15435 15727 15728 15432	1

Uudet jaksot

username	behaviour
myyntisihteeri	15432 15433 15434 15435 15727
myyntisihteeri	15433 15434 15435 15727 15728
myyntisihteeri	15434 15435 15727 15728 15727
myyntisihteeri	15435 15727 15728 15727 15728

ALARMS (käyttäytymisjaksot, jotka eivät vastaa profiilin jaksoja)

timestamp	username	session	behaviour
2.7.2002 16:44:53	myyntisihteeri	213	15434 15435 15727 15728 15727
2.7.2002 16:44:53	myyntisihteeri	213	15435 15727 15728 15727 15728

Esimerkissä käyttäjän toiminta ei poikkea opetusvaiheessa tallennetusta toimintaprofiilista, mutta käyttäjä suorittaa toimintoja järjestyksessä, jota vastaavaa ei ole tallennettu kyseisen käyttäjän käyttäytymisprofiiliin. Poikkeavat toimintojen jaksot tallennetaan poikkeavien toimintojen lisäksi tauluun ALARMS, josta on johdettu useita näkymiä eri tarkoituksiin ja poikkeamien tarkastelun tehostamiseen. Esimerkiksi näkymä ACTION_ALARMS näyttää vain käyttäjän poikkeavista toimista syntyneet hälytysmonikot ja BEHAV_ALARMS puolestaan käyttäytymisen perusteella syntyneet. Näkymä ACTION_ALARMS_DIST näyttää vain yksilölliset toiminnan poikkeamat, jolloin näytettävien poikkeamien määrää voidaan vähentää (LIITE 17).

Prototyyppijärjestelmään rakennettujen näkymien lisäksi hälytystiedoista voitaisiin muodostaa valikoituja joukkoja useiden eri lisämääreiden perusteella. Poikkeavasta käyttäytymisestä tallennettuja tietoja voitaisiin käyttää väärinkäytön tunnistuksessa esimerkiksi laskemalla poikkeamien ja kaikkien kyseisen istunnon jäljitystietojen määrästä poikkeamien suhteellinen osuus jonkin istunnon aikana. Näin tietoturvasta vastaava henkilö saisi tarkasteltavakseen sopivan näkymän kautta vain ne käyttäytymis-

poikkeamat, joita vastaavan istunnon aikana poikkeamia on enemmän kuin näkymälle asetettu raja-arvo. Tällöin pienet poikkeamat toimintojen suoritusjärjestyksessä eivät turhaan aiheuta hälytyksiä. Haluttaessa tarkastetut ja hyväksytyt käyttäytymisjaksot voitaisiin siirtää helposti profiileihin, jotta vastaavanlainen yhdistelmä ei aiheuta väärää hälytystä enää jatkossa. Myös toiminnan hyväksytyt poikkeamat voidaan opettaa toimintaprofiiliin. Tällöin on tosin huomioitava, että kyseessä olevaa toimintaa vastaavat käyttäytymisjaksot edeltävine ja seuraavine toimintoineen täytyy tallentaa myös käyttäytymisprofiiliin. Näin voidaan toteuttaa esimerkiksi Lanen (1998) tutkimuksessa tärkeäksi katsottu jatkuva oppiminen. Asiantuntijan valvonnassa ja ohjaamana suoritettu lisäopettaminen ei kuitenkaan altista järjestelmää niin helposti väärien mallien oppimisella kuin täysin automatisoitu järjestelmä.

5.7 Yhteenveto

Prototyyppijärjestelmä on rakennettu käyttäen hyväksi Oracle 9i –tietokannanhallinta-järjestelmän ominaisuuksia ja se on toteutettu täysin tietokannanhallintajärjestelmän sisällä. Jäljitystietojen kerääminen on toteutettu kolmella eri osajärjestelmällä: vakiojäljityksellä, hienojakoisella jäljityksellä sekä herättimillä. Kaikki tieto kootaan tausta-ajona ajettavilla tallennetuilla proseduureilla yhteen tietokannan tauluun, jonka attribuutit vastaavat kerättäviä jäljitystietojen ominaisuuksia. Taulun jäljitystietoja käyttävät hyväkseen opetusvaiheen normaalitoiminnan ja normaalikäyttäytymisen profiileja rakentavat proseduurit sekä tunnistamisvaiheen vastaavat poikkeamia etsivät proseduurit. Käyttäjän toimet mallinnetaan käyttäjänimen, toiminnan tunnuksen, taulukaavan, kohteen ja attribuutin nimen yhdistelmänä sen mukaan, mitkä tiedot kustakin tapahtumasta voidaan tallentaa. Käyttäytyminen puolestaan mallinnetaan tallennettujen toimien järjestettyinä jaksoina. Tiedot toiminnan ja käyttäytymisen poikkeamista tallennetaan omaan tauluunsa, jonka tietoja voidaan käyttää käyttäjän toimien tarkasteluun, vastatoimiin tai tarvittaessa profiilien jatkuvaan opettamiseen.

Seuraavassa luvussa on käsitelty tarkemmin prototyyppijärjestelmän rakentamisesta opittuja asioita sekä analysoitu prototyyppiä suhteessa aiempaan tutkimukseen. Myös prototyypin vahvuuksia ja heikkouksia on tarkasteltu.

6 KONSTRUKTION ANALYYSI JA JOHTOPÄÄTÖKSET

Prototyyppijärjestelmän konstruktion avulla saatiin paljon kokemuksia väärinkäytöntunnistusjärjestelmän rakentamisesta relaatiotietokannanhallintajärjestelmän yhteyteen. Myös aiemman tutkimuksen epäselviksi jättämät asiat saatiin selvitettyä ja muussa tutkimuksessa esitettyjä malleja yhdistettiin prototyyppijärjestelmän tunnistamisen monipuolistamiseksi.

Poikkeamaperusteista väärinkäytön tunnistamista koskevat julkaistut tutkimukset käsittelevät pääasiassa normaalitoimintaa tai –käyttäytymistä kuvaavien mallien muodostamista ja poikkeamien tunnistamista. Jäljitystietojen kerääminen järjestelmästä on usein ohitettu muutamalla maininnalla ja mikäli jäljitystietojen muodostamista ja tallentamista on käsitelty, se on ollut usein hyvin ylimalkaista ja epätarkkaa. Esimerkiksi Chung ym. (1999) on suunnitellut hienojakoisen tunnistamisen mallin, joka seuraa myös SELECT-lauseita attribuuttitasolla. Kuitenkaan tutkijoiden käyttämässä tietokannanhallintajärjestelmän versiossa tällaisten jäljitystietojen kerääminen ei ole mahdollista järjestelmän omilla vakiojäljitysominaisuuksilla. Joko tahallisesti tai vahingossa tämä oli kuitenkin jätetty mainitsematta. Samoin tietokantojen väärinkäytöntunnistamista koskevassa Ingriswangin ym. (2001) tutkimuksessa SELECT-lauseiden attribuuttitason jäljittäminen oli toteutettu epätarkasti dokumentoidulla kannan ulkopuolisella ohjelmalla.

Prototyyppijärjestelmää rakennettaessa pyrittiin selvittämään kunkin väärinkäytön tunnistamisen osa-alueen perusteet tyypillisessä kaupallisessa relaatiotietokannanhallintajärjestelmässä. Aiemmin julkaistun tutkimuksen pohjalta oletettiin, että jäljitystietojen kerääminen tietokannanhallintajärjestelmästä olisi melko suoraviivaista ja hoidettavissa järjestelmän omilla vakiojäljitysominaisuuksilla. Tämän vuoksi jäljitystietojen keräämistä alettiin suunnitella Oracle 8 –tietokannanhallintajärjestelmään eli samaan versioon, jolla aiempien tutkimusten konstruktiot oli rakennettu. Kävi kuitenkin ilmi, ettei tämä versio tukenut edellä mainittua SELECT-lauseiden hienojakoista jäljitystä, jolloin käyttöön otettiin uudempi versio 9i, jossa kyseinen ominaisuus oli.

Prototyyppijärjestelmässä jäljityksen lähtökohtana oli, että mikä tahansa tietokannan tapahtuma pystyttäisiin jäljittämään tarvittaessa sille luonnollisesti ominaisella tarkkuudella. Toinen lähtökohta oli, että koko järjestelmä pystyttäisiin toteuttamaan tietokannanhallintajärjestelmän sisällä, jolloin kyetään täysin hyödyntämään relaatiotietokannan kykyä käsitellä ja tallentaa tietoa. Jäljitystietoa ei tällöin myöskään tarvitse siirtää tietokannasta pois. Pahimmassa tapauksessa verkon kautta siirrettävä jäljitystieto saattaa joutua vääriin käsiin. Tietokannan ulkopuolista ohjelmointia ei välttämättä tarvita myöskään siksi, että yleisimmät kaupalliset tietokannanhallintajärjestelmät, kuten esimerkiksi Oracle ja DB2 tukevat tallennettuja proseduureja, joiden ohjelmointikieli sisältää kaikki kontrollirakenteet sekä tietojen käsittelyyn tarvittavat kirjastofunktiot.

6.1 Prototyyppijärjestelmän kokeileminen

Koska pelkän rakentamisen kokeilemisen lisäksi haluttiin kokeilla myös valmiin prototyyppijärjestelmän toimintaa, järjestettiin pienimuotoinen koe. Molempien esimerkkitaulukaavojen taulut ja niiden attribuutit asetettiin jäljittämään käyttäjien toimia, kuten edellä on kuvattu. Herättimet asetettiin jäljittämään INSERT- ja DELETE-lauseita taulutasolla ja UPDATE-lauseita attribuuttitasolla. Hienojakoisella jäljityksellä tallennettiin SELECT-lauseet attribuuttitasolla. Lisäksi myös Oraclen vakiojäljitys asetettiin päälle niiden toimintojen osalta, joita prototyyppijärjestelmä seuraa.

Jäljityksen asettamisen jälkeen tietokannan jokaisella kuvitteellisella käyttäjätilillä (TAULUKKO 4) tehtiin viisikymmentä toimintoa, jotka sisälsivät tietojen selaamista, päivittämistä, lisäämistä ja poistamista. Suoritettavat toimet valittiin se mukaan, mitä kunkin käyttäjän normaalitoimintaan käyttäjät kuvaavan taulukon mukaan kuuluu. Toimintoja suoritettiin satunnaisessa järjestyksessä. Käyttäjien normaalitoiminnan ja –käyttäytymisen profiileja muodostettiin taustalla koko ajan järjestelmää käytettäessä.

Tunnistamisvaiheeseen siirryttäessä profiileja rakentavien proseduurien ajastukset poistettiin tausta-ajosta ja toimintaprofiilien yksilöllisten tietojen määrät laskettiin

omalla proseduurillaan. Tämän jälkeen tunnistavat proseduurit asetettiin tausta-ajoon, jolloin ne alkoivat verrata uusia jäljitystietoja muodostettuihin profiileihin. Uusia jäljitystietoja muodostettiin jälleen toimimalla käyttäjien tileillä siten, että kymmenen toiminnon sarjasta kaksi ei käyttäjätaulukon mukaan kuulunut käyttäjien normaalitoimintaan. Yhdellä käyttäjistä kokeiltiin myös attribuuttitason jäljityksellä havaittavia poikkeamia, joita ei taulukkoon oltu suunniteltu.

Prototyyppijärjestelmän kokeilun tuloksia on käsitelty jäljempänä kunkin järjestelmän pääosan kohdalta erikseen. Seuraavassa on tarkasteltu jäljitystietojen keräämisestä saatuja tuloksia järjestelmän rakentamisen sekä kokeilun näkökulmasta.

6.2 Jäljitystietojen keräämisestä tehtyjä havaintoja

Jäljitystietojen kerääminen ratkaistiin prototyyppijärjestelmässä kolmella eri osajärjestelmällä, joiden avulla kaikki tietokannan toiminta voidaan haluttaessa jäljittää sopivalla tarkkuudella. Oraclen vakiojäljitystä käyttämällä saadaan tieto kaikista niistä tietokannan toiminnoista, joista ei tarvitse tallentaa attribuuttitason jäljitystietoa. Attribuuttitason jäljitys voidaan asettaa UPDATE-lauseille herättimin ja SELECT-lauseille erityisellä Oraclen DBMS_FGA-ohjelmapaketilla, joka on tarkoitettu attribuuttitason hienojakoiseen jäljittämiseen. INSERT- ja DELETE-lauseiden jäljittäminen voidaan hoitaa haluttaessa joko vakiojäljityksellä tai herättimillä. Prototyyppijärjestelmässä valittiin herättimien käyttö niiden kokeilemiseksi, vaikka vakiojäljityksen käyttö näille toiminnoille olisi ollut todennäköisesti järjestelmän suorituskyvyn kannalta edullisempaa. Herättimille ja hienojakoiselle jäljitykselle voidaan lisäksi asettaa mielivaltaisen monimutkaisia ehtoja, joiden perusteella jäljitystieto muodostetaan tai jätetään muodostamatta. Tällaisia ehdollisia herättimiä ja hienojakoista jäljitystä voitaisiin käyttää Ingriswangin ym. (2001) esittämän järjestelmän kaltaisiin sääntöihin, joilla suojattaisiin erityisen tärkeitä tietoja. Säännöt voisivat poikkeamien tunnistuksen lisäksi kirjoittaa ehtojen täyttyessä hälytyksen suoraan järjestelmänvalvojan tarkasteltavaksi.

Prototyyppijärjestelmässä jäljitysasetukset valittiin siten, että kaikkien käyttäjien toimia valvottiin samalla tavoin. Kuitenkin haluttaessa järjestelmällä olisi täysin mahdollista

asettaa jäljitys siten, että jokaisen käyttäjän toimintaa seurattaisiin yksilöllisesti. Kaikki jäljityksen osajärjestelmät tukevat ehtoja, joilla jäljitys voidaan asettaa tarkkailemaan vain tietyn käyttäjän tai käyttäjäryhmän toimintaa. Tällöin tosin jäljitysasetusten määrittelytyö ja tietokantaan tallennettavien herättimien ja sääntöjen määrä moninkertaistuisi.

Prototyyppijärjestelmää rakennettaessa havaittiin myös joitakin Oracle-tietokannanhallintajärjestelmän erityisiä ominaisuuksia, jotka rajoittivat järjestelmän toteutusta. Koska vakio- ja hienojakoisen jäljityksen tiedot tallentuvat SYS-taulukaavassa sijaitseviin tauluihin, joille järjestelmä ei anna asettaa herättimiä, oli löydettävä toisenlainen tapa siirtää uudet jäljitystiedot käsittelyyn. Jäljityksen lokitauluja ei rajoitusten vuoksi myöskään voinut siirtää toiseen taulukaavaan herättimien asettamiseksi. Ainoaksi mahdollisuudeksi jäi ajastaa tallennetut proseduurit, jotka tietyin väliajoin tarkastavat lokit uusien jäljitystietojen varalta. Opetusvaiheessa tällä Oraclen rajoituksella ei juurikaan ole merkitystä, sillä profiilit voitaisiin yhtä hyvin muodostaa käsin käynnistettävällä proseduurilla kerta-ajona. Tunnistamisvaiheessa täytyy kuitenkin olla jokin tapa reagoida ainakin lähes reaaliaikaisesti jäljityslokeihin muodostuviin uusiin jäljitystietoihin, jotta mahdollisiin väärinkäyttötapauksiin voidaan puuttua ajoissa.

Herättimen avulla uuden jäljitystiedon muodostumisen jälkeen käynnistettävä poikkeamien tunnistus ei muutenkaan prototyyppijärjestelmästä saatujen kokemusten perusteella olisi paras mahdollinen ratkaisu. Herättimen käyttäminen aiheuttaa jäljitystiedon muodostumiseen johtavan tietokantasovelluksen tapahtuman sekä jäljitystiedon kirjaamisen yhdistymisen transaktioksi. Tällöin jäljitystiedon tallentaminen pitää suorittaa loppuun asti, ennen kuin tietokantasovelluksen tapahtuma vahvistuu. Mahdollinen tallennusvirhe voisi pahimmassa tapauksessa estää tietokantasovelluksen käytön ja normaalisti toimiessaankin tällainen järjestely hidastaisi sovelluksen käyttöä tarpeettomasti. Kun uudet jäljitystiedot siirretään keskitettyyn lokiin tausta-ajona, jäljitystietojen siirtäminen ei häiritse tietokantasovelluksen käyttöä.

Kokeiltaessa prototyyppiä kuvitteellisilla käyttäjillä, kaikki jäljitettäviksi asetetut tiedot tallentuivat järjestelmän rakenteen mukaisesti joko aluksi erillisiin lokeihin ja sitten keskitettyyn lokiin tai suoraan keskitettyyn lokiin. Jokaisen suoritetun toiminnon

jälkeen valvottiin, että jäljitystieto oli asianmukaisesti tallentunut keskitettyyn jäljitystietotauluun. Valvonnan tuloksena todettiin kaikkien suoritettujen toimintojen tulleen asianmukaisesti jäljitetyiksi. Vaikkakaan pienimuotoinen koejärjestely ei pystynyt osoittamaan prototyyppijärjestelmän jäljitystietojen keräämisen tehokkuutta, voitiin kuitenkin todentaa sen toimivan aiotulla tavalla ja keräävän luotettavasti jäljitystiedot, jotka se oli asetettu keräämään.

Prototyyppijärjestelmällä saatiin selvitettyä muussa tutkimuksessa vähän käsiteltyä jäljitystietojen keräämistä tietokannanhallintajärjestelmästä. Vaikka Oraclen erityisominaisuuksista saadut kokemukset eivät ole tietysti suoraan sovellettavissa muihin relaatiotietokannanhallintajärjestelmiin, saatiin prototyyppijärjestelmästä tärkeää tietoa siitä, millaisia ongelmia jäljitystietojen keräämisessä ja käsittelyssä saattaa esiintyä. Parhaaksi saavutukseksi prototyyppijärjestelmän jäljitystietojen keräämisessä voidaan katsoa se, että käytännössä kaikista tietokannanhallintajärjestelmän toiminnoista voidaan haluttaessa kerätä tietoa semanttisesti sopivalla tarkkuudella. Poikkeamaperusteisen väärinkäytöntunnistamisen peruslähtökohta täytyy olla monipuolinen jäljitystietojen kerääminen. Tietokantajärjestelmässä on useita erityyppisiä toimintoja ja tarkkuustasoja, joista tietoa pitää tallentaa. Prototyyppijärjestelmässä kutakin tietokantasovellusta varten voidaan riittävän monipuolisen jäljitystietojen keräysjärjestelmän avulla räätälöidä juuri sopivat asetukset, joilla saadaan luotua parhaiten käyttäjien normaalitoimintaa ja –käyttäytymistä kuvaavat profiilit.

Luvussa 3 käsiteltyjä vahvuuksien ja heikkouksien vertailuperusteita tarkasteltaessa voidaan todeta, että prototyyppijärjestelmässä on onnistuttu helpottamaan järjestelmän konfigurointia jäljitystietojen keräämisen osalta Oraclen vakiona tarjoamiin välineisiin verrattuna. Jäljitystietojen keräysasetusten tekemiseen on prototyyppijärjestelmässä ohjelmoitu joitakin apuvälineitä vähentämään komentojen kirjoittamista, mutta valmis järjestelmä vaatisi tuekseen vielä graafisella käyttöliittymällä varustetun apuvälineen, josta jäljitettävät kohteet näkisi ja jolla uusia kohteita voisi lisätä luetteloista valitsemalla. Resurssikäytön ongelman näkökulmasta olisi mielenkiintoista tutkia, kuinka paljon yhden jäljitystiedon kerääminen varaa tietokannanhallintajärjestelmää ja miten monista kohteista ja miten hienojakoisesti jäljitystietoja voi kerätä ilman, että kerääminen häiritsee merkittävästi muuta toimintaa. Myös muu väärinkäytöntunnista-

misjärjestelmän toiminta täytyy olla käyttäjälle melko näkymätöntä. Seuraavassa kohdassa on käsitelty prototyyppijärjestelmän profiilien muodostamisen periaatteita ja käytännön ratkaisuja.

6.3 Normaalitoimintaa ja –käyttäytymistä kuvaavien profiilien rakentamisesta

Kuten edellä käsiteltiin, voidaan Oracle-tietokannanhallintajärjestelmän tallennettujen proseduurien ohjelmoinnissa käytettävällä PL/SQL-kielellä ohjelmoida kaikki tarpeelliset kontrollirakenteet. Lisäksi käytettävissä ovat kaikki jäljitystiedon käsittelyssä tarvittavat kirjastofunktiot sekä tiedot tietokannasta tietohakemiston tauluissa. Tämän vuoksi prototyyppijärjestelmällä ei katsottu olevan tarpeellista todistaa minkään tietyn profiilien muodostamisen mallin rakentamisen mahdollisuuksia, sillä periaatteessa jäljitystiedoista voidaan Oraclen ohjelmointimahdollisuuksilla jalostaa millaisia profiileja tahansa.

Prototyyppijärjestelmän profiilien rakentamisessa mukailtiin tapaa, jolla Oracle-tietokannanhallintajärjestelmä kerää jäljitystietoja. Vakiojäljityksen sekä hienojakoisen jäljityksen tuottamat tiedot tallennetaan omiin lokitauluihinsa, joissa jokaiselle tiedolle on varattu oma sarakkeeseensa. Tätä tallentamisen tapaa tai taulujen rakennetta ei voi muuttaa. Tämän vuoksi profiilit valittiin rakennettavaksi jäljitystietojen rakennetta vastaaviksi, tosin suppeammiksi, sillä profiileihin ei haluttu sisällyttää kaikkea mahdollista tietoa, joka jäljityksellä olisi ollut hankittavissa. Jäljitystietojen tapaan tallennettaessa profiilien käsittely on yksinkertaista, eikä ominaisuuksien kiinnittämisen näennäinen ongelmakaan tosiasiassa haittaa, sillä ennen opetusvaiheen aloittamista profiilien rakentamisessa käytettävät jäljitystietojen ominaisuudet täytyy kuitenkin päättää.

Tuotantokäyttöön tulevassa järjestelmässä tulisi tietysti voida valita, mitä ominaisuuksia jäljitystietojen ja profiilien vertailussa käytetään, mutta prototyypissä tällä ei katsottu olevan merkitystä. Voitaisiin esimerkiksi ajatella, että järjestelmä voitaisiin asettaa vertailemaan pelkästään käyttäjien suorittamia toimintoja ilman, että toiminnan kohteista tallennetaan tietoa. Toinen mahdollisuus supistaa profiileja olisi jättää attribuuttitason tietojen tallennus pois. Tällaisen ominaisuuksien valinnan ei kuitenkaan

tarvitse vaikuttaa järjestelmän perustana oleviin tietoa tallentaviin tauluihin, vaan ainoastaan niitä käsittelevään ohjelmistoon. Lisäksi taulujen ja tietoja käsittelevien proseduurien muuttaminen ennen opetusvaiheen aloittamista on verrattain helppoa, jos halutaan lisätä seurattavia jäljitystiedon ominaisuuksia. Sopivia lisäominaisuuksia voisivat olla esimerkiksi diskreetisti mallinnettu aika tai paikka, josta käyttäjä toimintoja suorittaa.

Merkittävin saavutus prototyyppijärjestelmän profiilien muodostamisessa on selkeä jako käyttäjän toiminnan ja käyttäytymisen mallintamisessa. Pääosassa aiempaa tutkimusta ei ole juuri tehty eroa käyttäjän toiminnan ja käyttäytymisen tallentamisen välillä. Kuitenkin näkemysero eri tutkijoiden kesken on selvästi havaittavissa. Poikkeamaperusteisessa tutkimuksessa on perinteisesti pyritty tunnistamaan käyttäjä käyttäytymisen perusteella esimerkiksi tallentamalla tilastollisesti järjestelmän toimintaa. Kuitenkin uudemmassa tutkimuksessa ja varsinkin tietokannanhallintajärjestelmille suunnitelluissa malleissa pyritään tunnistamaan millainen toiminta käyttäjälle on normaalia, eikä niinkään pyritä identifioimaan käyttäjää. Esimerkiksi Chungin ym. (1999) malli on selvästi saanut vaikutteita käyttöoikeuksien hallinnasta, kun taas esimerkiksi Anderson ym. (1995) malli on edelleen selvästi velkaa tilastotieteelle ja biometrisille menetelmille.

Prototyyppijärjestelmän molemmat profiilit pitävät kaikkea oletusarvoisesti epäilyttävänä ja hyväksyvät normaaliksi vain sellaisen toiminnan, jonka ovat oppineet opetusvaiheessa esimerkkien perusteella. Käyttäjän toiminta mallinnetaan opetusvaiheen jäljitystietojen perusteella siten, että kustakin jäljitystiedosta tallennetaan tekijä, toiminta sekä kohde kolmella tasolla sen mukaan, millä tarkkuudella kyseisestä tapahtumasta voidaan tallentaa tietoa. Näin prototyyppijärjestelmän profiilit tallentavat edellä käsitellyn Denningin (1987) poikkeamaperusteisen mallin kolme tärkeintä jäljitystietojen ominaisuutta. Profiileihin tallennetaan vain yksilölliset tiedot, ja mikäli useampia samanlaisia jäljitystietoja esiintyy, niiden määrä lasketaan. Määrän perusteella voidaan haluttaessa esimerkiksi karsia profiilista harvinaisia jäljitettyjä tapahtumia. Prototyyppijärjestelmässä profiilien tietoja ei kuitenkaan karsittu.

Toimintaprofiilin avulla voidaan tarkalla tasolla erottaa, mikä käyttäjän toiminnasta tietokannan suhteen on aiemmin havaitun perusteella normaalia ja mikä poikkeavaa. Profiilit on kuitenkin pidetty tietokannanhallintajärjestelmän tasolla, eikä niihin ole sekoitettu sovellustason ominaisuuksia, kuten Chungin ym. (1999) mallissa annettiin ymmärtää. Olisi epäkäytännöllistä tallentaa opetusvaiheessa esimerkiksi yksittäisen pankkitoimihenkilön käsittelemien tapahtumien sisältöä sen päättelemiseksi, onko tunnistusvaiheessa syötetty hyvitettävän pankkitilin numero aiemmin tunnettu. Sen sijaan tällaiseen sovellustason väärinkäytön tunnistamiseen voitaisiin käyttää Ingriswangin ym. (2001) esittämiä sääntöjä, joilla voitaisiin esimerkiksi määritellä, ettei pankkitoimihenkilö voi hyvittää tiliä, jonka hän itse omistaa. Yhtä hyvin opetusvaiheessa voitaisiin laskea normaalien tilisiirtojen summien keskiarvo ja keskipoikkeama, jonka perusteella voitaisiin tutkia, ovatko pankkitoimihenkilön kirjaamat siirrot suuruudeltaan ja määrältään epänormaaleja. Tällaiset erityiset sovellusta koskevat säännöt voisivat täydentää prototyyppijärjestelmän kaltaisen poikkeamaperusteisen järjestelmän tunnistamista ja ne voisivat toimia poikkeamaperusteisen järjestelmän rinnalla kirjoittaen tarvitsemansa vertailutiedot opetusvaiheessa omiin lokeihinsa.

Käyttäytymisprofiilin perusajatus on adoptoitu Lanen (1998) ja Hofmeyerin ym. (1998) tutkimuksista, joissa käyttäjän käyttäytyminen mallinnetaan toimintojen ajallisesti järjestettyinä vakiomittaisina sarjoina. Hofmeyer ym. (1998) olivat käyttäneet sarjojen muodostamisessa ainoastaan yksinkertaisia järjestelmäkutsuja, joiden ansiosta profiilit olivat kooltaan hyvin pieniä ja käsittely tehokkasta. Tutkijat kuitenkin totesivat, että malli ei välttämättä anna kovin hyvää kuvaa siitä, mitä järjestelmässä itse asiassa tapahtuu. Lane (1998) puolestaan on rakentanut profiilinsa samalla periaatteella monipuolisemmista tiedoista, jonka vuoksi profiilit ovat kooltaan isompia. Kuitenkaan järjestelmän tunnistamisen tarkkuus ei ollut paras mahdollinen, sillä käyttäjän käyttäytymistä mallintaville poikkeamaperusteisille järjestelmille tyypillisesti profiilit eivät pysty koskaan kuvaamaan käyttäytymistä täydellisesti. Järjestelmä aiheuttaakin sen vuoksi paljon vääriä hälytyksiä. Lane (1998) ratkaisi ongelman laskemalla edellä kuvatulla tavalla porrastetun vastaavuussuureen avulla ja lisäksi tasoitti vastaavuudet laskemalla keskiarvon useasta jäljitystapahtumasta. Ilmeistä on, ettei tällainen poikkeamien tunnistaminen pysty antamaan tietoa mahdollisesta väärinkäytöstä

reaaliaikaisesti, vaan se voi vasta pitkällä aikavälillä ilmoittaa käyttäjän toiminnan muuttumisesta.

Prototyyppijärjestelmässä käyttäytymisprofiilien muodostamisessa on käytetty hyväksi toimintaprofiiliin tallennettuja tietoja käyttäjän toiminnasta. Käyttäytymisprofiilin ajallisesti järjestettyinä jaksoina tallennetaan vain viitteitä toimintaprofiiliin, jolloin profiili vie mahdollisimman vähän tallennustilaa. Lisäksi jaksojen osat identifioivat tekijän, toiminnan ja sen kohteen yksityiskohtaisesti, jolloin käyttäytymisen mallintamisen perusteet ovat tarkat. Käyttäytymisprofiilien jaksot on kuitenkin muodostettu Lanen (1998) ja Hofmeyerin ym. (1998) profiilien tapaan liikuttamalla vakiomittaista ikkunaa jäljitystietojen joukon yli. Jäljitystietojen sarjoja ei kuitenkaan tallenneta sellaisenaan, vaan jakso muodostetaan jäljitystietoja vastaavista toimintaprofiilin tunnusnumeroista. Näin prototyyppijärjestelmässä on yhdistetty käyttäjän antamien komentojen ajalliseen järjestykseen perustuva käyttäytymisen mallintaminen hyvin hienojakoiseen käyttäjän toiminnan mallintamiseen. Käyttäytymisen mallintamisessa on lisäksi pyritty mahdollisimman tiiviiseen esitykseen.

Attribuuttitason jäljitys tallentaa prototyypin kaltaisessa esimerkkien pohjalta oppivassa instanssiperusteisessa järjestelmässä useita monikkoja normaalitoiminnan profiiliin, jos käyttäjän toimet koskevat useita taulun attribuutteja. Saattaa olla, että käyttäjä jossain tapauksessa viittaa kaikkiin taulun attribuutteihin, jolloin attribuuttitason jäljitys toiminnan jäljittämisen kannalta menettää merkityksensä. Jos järjestelmällä seurattaisiin ainoastaan käyttäjän toiminnan poikkeamia, profiilia voitaisiin tällöin tiivistää korvaamalla yksittäiset attribuuttitason profiilitiedot yhdellä taulutason tiedolla. Koska prototyyppijärjestelmä kuitenkin seuraa myös käyttäjän käyttäytymistä omassa profiilissaan, jonka perusteena on käytetty toimintaprofiilin tietoja, ei tällaista tiivistämistä voida käyttää. Käyttäytymistä mallinnettaessa on tärkeä erottaa tarkkaan, mihin attribuuttien osajoukkoon viittaus kulloinkin kohdistuu. Myöskään Chungin ym. (1999) esittämää periaatteellista ajatusta minimaalisesta usein esiintyvien ominaisuuksien joukosta ei voida, eikä ole tarpeellista soveltaa. Toimintaprofiilia rakentava proseduuri pitää huolta, ettei redundantteja toimintoja tallenneta lainkaan. Käyttäytymisprofiili huolehtii tämän jälkeen yksittäisten toimintojen ryhmittämisestä tapahtumiksi.

Prototyyppijärjestelmää koestettaessa valvottiin profiilien muodostamista koko ajan koekäyttäjien tunnuksilla toimittaessa, koska profiilit muodostettiin samanaikaisesti opetusvaiheen jäljitystietoja kerättäessä. Järjestelmä käsitteli kaikki jäljitystiedot asianmukaisesti ja samanlaisista toiminnan jäljitystiedoista profiiliin tallentui vain yksi kappale, jolloin profiileihin ei muodostunut redundanttia tietoa. Myös käyttäytymisprofiilit rakentuivat oikeellisesti keskitetyn jäljitystietotaulun ajallisen järjestyksen mukaan. Kehitettävää sen sijaan löydettiin eri jäljityksen osajärjestelmien tuottaman tiedon keskinäisen edeltävyyden hallitsemisesta. Koska Oraclessa tieto tapahtuman ajasta tallentuu vain sekunnin tarkkuudella, ei kellonaikaa voida yksin käyttää määräämään järjestystä. Kunkin osajärjestelmän sisällä järjestys on helppo päätellä sen mukaan missä järjestyksessä tiedot ovat muodostuneet. Sisäinen järjestys saadaan sekvenssien luomista identifioivista tunnuksista. Kuitenkin samaan kellonaikaan samalle käyttäjälle eri osajärjestelmiin muodostuneiden tietojen järjestystä ei prototyyppijärjestelmällä koetilanteessa pystytty selvittämään. Lisäksi viitattaessa yhdessä SQL-lauseessa useampiin saman taulun attribuutteihin, niiden jäljitystiedot tallentuivat aina satunnaisessa järjestyksessä. Toisaalta attribuuttitason järjestys on helppo vakioida vaikkapa käyttämällä hyväksi tietohakemiston järjestystä. Sen sijaan eri osajärjestelmien välinen tarkka järjestys vaatisi toimiakseen jonkin ylemmällä tasolla toimivan järjestelmän, joka määräisi järjestyksen. Tähän voitaisiin käyttää esimerkiksi Oraclen vakiojäljitystä, joka pystyy jäljittämään taulutasolle asti kaikki järjestelmän toiminnot. Herättimiltä ja hienojakoisesta jäljityksestä voitaisiin tämän jälkeen hakea tarkempi tieto viitatuista attribuuteista niille toiminnoille, joille tällainen tieto voidaan tallentaa.

Tarkasteltaessa luvussa 3 esitettyjä profiilien rakentamisvaiheen heikkouksia ja vahvuuksia, voidaan todeta, että prototyyppijärjestelmässä poikkeamaperusteisille järjestelmille tyypillisten heikkouksien aiheuttamat ongelmat onnistuttiin pitämään kohtuudella hallinnassa. Sekä toiminta- että käyttäytymisprofiilit veivät kohtuullisen vähän tallennustilaa. Toiminnan tallentamiseen tarvittiin vain muutamia kilotavuja käyttäjää kohden ja käyttäytymisen profiilit veivät tilaa muutamia kymmeniä kilotavuja. Profiilien käyttämä levytila kunkin käyttäjän osalta on esitetty oheisessa taulukossa (TAULUKKO 10). Levytilavaatimukset on laskettu profiiliin muodostuneiden monikoiden lukumäärän ja yhden profiilimonikon viemän tallennustilan keskiarvon mukaan.

TAULUKKO 10. Toiminta- ja käyttäytymisprofiilien tallennuksen vaatima levytila käyttäjittäin.

käyttäjä	monikkoja toimintaprof.	tilantarve toim. prof. (ka 54 tavua)	monikkoja käyttäytymisprof.	tilantarve käyt. prof. (ka 60 tavua)
myyntipäällikkö	91 kpl	4914 tavua	765 kpl	45900 tavua
myyntisihteeri	82 kpl	4428 tavua	317 kpl	19020 tavua
palkanlaskija	48 kpl	2592 tavua	1106 kpl	66360 tavua
toimitusjohtaja	87 kpl	4698 tavua	778 kpl	46680 tavua
varastomies	66 kpl	3564 tavua	320 kpl	19200 tavua
ylläpito	18 kpl	972 tavua	32 kpl	1920 tavua

Koska järjestelmän tehokkuuskysymykset eivät kuitenkaan olleet tutkielman pääasiallinen mielenkiinnon kohde, profiilien tallennusvaihtoehtojen tehokkuutta ei tutkittu tarkemmin. Oraclen tietohakemistot tarjoaisivat esimerkiksi mahdollisuuden tallentaa tietokannan käyttäjät, kohteet ja attribuutit niitä kuvaavin numerotunnuksin. Vakio- ja hienojakoinen jäljitys kuitenkin tallentavat kyseiset jäljitystietojen ominaisuudet merkkijonoina. Merkkijonoina tapahtuvan vertailun muunto numerotunnusperusteiseksi saattaisi tehostaa järjestelmää. Samoin profiilit voitaisiin tallentaa relaatiomallin mukaan siten, että esimerkiksi toiminta ja sen kohde tallennettaisiin erikseen ja tieto käyttäjistä, jotka suorittaisivat toimintoja kyseisille kohteille tallennettaisiin erikseen. Tallennettavien monikoiden määrä laskisi tällöin ja redundanttia tietoa ei tallennettaisi, mutta sekä numerotunnusten käyttö että tiedon jakaminen eri tauluihin lisäisi tarvittavien liitosten määrää tietoa käsiteltäessä. Olisi mielenkiintoista tutkia, missä määrin erilaisin rakenneratkaisuin on mahdollista tehostaa poikkeamaperusteisen tunnistuksen tehokkuutta relaatiotietokannanhallintajärjestelmässä ja millainen olisi optimaalinen järjestelmä suorituksen tehokkuuden ja tallennustilan edullisen käytön suhteen.

6.4 Havaintoja poikkeamien tunnistamisesta

Jäljitystietojen keräämisen konfiguroinnin ja profiilien rakentamisen jälkeen poikkeamien tunnistus on prototyyppijärjestelmässä varsin suoraviivaista. Poikkeamien tunnistamiseen käytetään samaa perusajatusta kuin uusien jäljitystietojen profiileihin tallentamiseen. Toimintaprofiilia verrataan uusiin jäljitystietoihin etsimällä vastineetto-

mia tietueita SQL-kyselyllä. Liitäntä tehdään käyttäjänimen, toiminnan numerotunnuksen, taulukaavan, kohteen ja attribuutin nimen perusteella. Jos tunnistusvaiheen kaikkia kerrallaan vertailtavia jäljitystietoja vastaavat tiedot löytyvät normaalitoiminnan profiilista, on käyttäjän toiminta normaalia. Mikäli vastineettomia uusia jäljitystietoja löytyy, ne siirretään hälytystauluun tarkempaa tutkimista varten. SQL-kyselyä käytettäessä tunnistamisen tehokkuus on aina optimaalinen, koska tietokannanhallintajärjestelmä automaattisesti optimoi kyselyjen suorituksen.

Koska käyttäjän toiminnan tunnistamisen periaate on yksinkertainen ja käytännössä hienojakoisen käyttöoikeuksienhallintajärjestelmän kaltainen, poikkeamien tunnistamista voidaan pitää hyvin luotettavana väärien negatiivisten osalta. Järjestelmähän pitää oletuksena kaikkea sille tuntematonta toimintaa epäilyttävänä. Vääriä positiivisia hälytyksiä tällaisessa järjestelmässä saattaa puolestaan aiheuttaa käyttäjän toiminnan muuttuminen opetusvaiheen jälkeen tai jonkin normaalin työtehtävän ajoittuminen opetusvaiheen ulkopuolelle. Väärät positiiviset eivät kuitenkaan prototyypin kaltaisessa järjestelmässä ole pysyvä ongelma, sillä yksinkertaisesti esimerkkien perusteella oppiva järjestelmä on hyvin muuntumiskykyinen ja joustava jatkuvan oppimisen kannalta. Varsinkin toimintaprofiilien päivittäminen on helppoa, sillä väärän hälytyksen aiheuttama tallentamaton toiminto voidaan heti tallentaa profiiliin. Käyttäjän normaalitoiminnan profiilia voidaan päivittää yksinkertaisesti puuttuvat tiedot lisäämällä, eikä profiilia tarvitse rakentaa kokonaan uudelleen. Kuitenkin toimintaprofiilin lisäksi vastaavat jaksot tulee lisätä myös käyttäytymisprofiiliin. Prototyyppijärjestelmään tätä ominaisuutta ei ole toteutettu, mutta tarkoitukseen kävisi proseduuri, joka hakee jäljitystiedoista järjestyksessä jakson pituutta vastaavan määrän uutta toimintoa edeltäviä ja seuraavia jäljitystietoja ja luo niistä käyttäytymisprofiiliin uudet jaksot.

Käyttäjän toimintaa tunnistettaessa vääriä positiivisia hälytyksiä voitaisiin haluttaessa rajata myös esimerkiksi käyttäen hyväksi Chungin ym. (1999) esittämää rakenteellisen etäisyyden käsitettä. Prototyyppijärjestelmään ohjelmoidulla proseduurilla REAL_DIST voitaisiin laskea poikkeavan toiminnan kohteen etäisyys lähimmästä normaalista toiminnan kohteesta ja tietyn asetetun raja-arvon perusteella näytettäviä hälytyksiä voitaisiin karsia tai asettaa tärkeysjärjestykseen. Kuten edellä jo käsiteltiin, kokonaan toinen kysymys on, kuinka paljon pelkän rakenteellisen etäisyyden perusteella voidaan

päätellä poikkeaman merkittävyydestä. Esimerkiksi samassa taulussa sijaitsevien henkilön nimen ja henkilötunnuksen rakenteellinen välimatka on pieni, mutta tietoturvan kannalta niiden luottamuksellisuudessa on merkittävä ero.

Prototyyppijärjestelmää koestettaessa käyttäjän normaalitoiminnan tunnistaminen toimi varsin luotettavasti. Tunnistamisvaiheessa järjestelmä salli oikein kaiken opetetun ja käyttäjätaulukon mukaan normaaliksi luokitellun toiminnan sekä tunnisti oikein epäilyttäviksi koekäytössä tahallaan käyttäjän normaalitoiminnan profiilin ulkopuolelle tehdyn käytön. Luokittelu toimi oikein kaikille käyttäjille, toiminnoille ja kohteille. Tunnistamisvaiheen kokeiluissa seurattiin jäljitystietojen muodostumista keskitettyyn jäljitystietotauluun sekä hälytysten muodostumista niille varattuun tauluun. Järjestelmään ei myöskään toiminnan tunnistamisen vuoksi syntynyt juurikaan viivettä, vaan hälytykset epäilyttävästä toiminnasta tulivat lähes reaaliaikaisesti.

Poikkeavan käyttäytymisen tunnistamisesta saatiin prototyyppijärjestelmästä kohtuullisen vähän luotettavia tuloksia, sillä luotettava käyttäytymisen tunnistaminen olisi selvästi vaatinut laajempaa koeaineistoa tai erityisen tarkkaan tehtyä suunnitelmaa siitä, missä järjestyksessä toiminnot kulloinkin suoritetaan. Tietokantasovellusta käytettäessä pyrittiin järjestelmällisyyteen ja toimimaan kohtuudella samalla tavoin kuin opetusvaiheessakin. Järjestelmä tunnisti normaaliksi käyttäytymiseksi suurimman osan uusista käyttäytymistä kuvaavista jaksoista, mutta muodosti vääriä hälytyksiä kuitenkin 1 - 4 käyttäjää kohti. Tämä johtuu siitä, että prototyyppijärjestelmässä seurattiin vain absoluuttisesti poikkeavia jaksoja, eikä poikkeamia laskettu esimerkiksi jakson osien perusteella porrastetusti ja tämän jälkeen raja-arvoon vertaamalla. Lisäksi opetusvaiheen tapahtumien pieni määrä ei tuottanut tarpeeksi monipuolista käyttäytymisprofiilia, jotta kaikki käyttäjälle tyypilliset tavat tehdä työtä olisivat tulleet tallennetuiksi. Yhden käyttäjän osalta kokeiltiin opetusvaiheen jäljitystietojen järjestyksen täsmällistä seuraamista tunnistusvaiheessa, jolloin vääriä positiivisia hälytyksiä ei muodostunut. Prototyyppijärjestelmässä käyttäytymisen tunnistamisen tarkkuus vaatii saatujen kokemusten perusteella siis huomattavasti pidempää opetusvaihetta, jonka aikana käyttäjän toimien suorittamisen kaikki tyypilliset järjestykset tulevat tallennetuiksi. Myöskään täysin absoluuttinen ja yksittäisen jakson perusteella tehty poikkeamien tunnistus ei ole tehokasta. Käyttäytymisen poikkeamien tunnistamisessa tulisi myös prototyypin

kaltaisessa järjestelmässä käyttää esimerkiksi Lanen (1998) esittämän kaltaista käyttäytymisjakson osien erillistä vertailua ja vastaavuusarvon laskemista. Vastaavuusarvoa tai useamman vastaavuusarvon keskiarvoa voitaisiin tällöin vertailla asetettuun normaalivastaavuuden minimiin. Asetettua minimivastaavuutta enemmän poikkeaville jaksoille voitaisiin myös asettaa jokin määrä, jonka ylittäminen yhden istunnon aikana aiheuttaisi tietojen kirjoittamisen hälytystauluun.

Luvussa 3 käsiteltyjen poikkeamaperusteisille järjestelmille tyypillisten heikkouksien ja vahvuuksien suhteen tarkasteltuna protyyppijärjestelmääkin vaivaa jonkin verran väärien positiivisten tunnistusten ongelma. Toimintaprofiilien aiheuttamista vääristä positiivisista voitaisiin saada tietoa vain todellista järjestelmää tutkimalla, mutta käyttäytymisprofiilien osalta ongelma on havaittavissa jo prototyyppijärjestelmässäkin. Selvä vahvuus on kuitenkin toimintaprofiilien periaate, jolla kaikki aiemmin tallentamattomasta poikkeava toiminta voidaan tunnistaa tarkasti ja täsmällisesti.

Myös käyttäytymisen osalta jatkuvan oppimisen periaate on melko helposti toteutettavissa. Edellä käsiteltiin, kuinka toimintaprofiilia päivitettäessä ja uutta toimintaa lisättäessä myös käyttäytymisprofiilia tulee päivittää. Tällöin uudet käyttäytymisjaksot tulee muodostaa tutkimalla järjestettyä jäljitystietojoukkoa jakson pituuden verran ennen ja jälkeen uutta toimintoa. Mikäli toiminta itsessään ei ole uusi, vaan järjestelmälle opetetaan vain uusia toimintojen suoritusjärjestyksiä, ne voidaan hyväksymisen jälkeen lisätä profiiliin suoraan hälytystaulusta siirtämällä. Tällöin uutta hälytystä ei synny käyttäjän suorittaessa toimintoja seuraavan kerran samassa järjestyksessä. Esimerkkien pohjalta oppivassa ja tunnistavassa järjestelmässä sekä normaalitoiminnan että normaalikäyttäytymisen profiilien päivittäminen on yksinkertaista verrattuna järjestelmään, jossa malli rakennetaan kiinteäksi, kuten puut tai neuroverkot.

Olisi mielenkiintoista järjestää useammille aitojen tietokantasovellusten käyttäjille koe, jossa pitkän opetusvaiheen ja tunnistusvaiheen tiedoista laskettaisiin, miten pitkä opetusvaihe suhteessa tietokantasovelluksen kokoon tarvittaisiin käyttäytymisen tunnistamiseen mahdollisimman vähillä väärillä negatiivisilla. Prototyyppijärjestelmällä pystyttiin osoittamaan toimintaprofiilin luotettava toiminta tunnistamisvaiheessa, mutta samalla todettiin, että käyttäytymisen mallintamiseen käytetty opetusvaiheen pituus oli

liian lyhyt. Toinen tärkeä tunnistamisvaiheeseen liittyvä tutkimuksen kohde olisi selvittää, miten paljon prototyypin kaltainen väärinkäytöntunnistamisjärjestelmä varaa relaatiotietokannanhallintajärjestelmän resursseja. Väärinkäytöntunnistamisjärjestelmän täytyy olla tietokantasovelluksen käyttäjälle näkymätön, eikä se saa merkittävästi hidastaa tarkkailemaansa järjestelmää.

6.5 Yhteenveto

Prototyyppijärjestelmän rakentamisesta ja kokeilusta saatiin arvokasta tietoa varsinkin jäljitystietojen keräämisen osalta. Aiemmassa tietokannanhallintajärjestelmien väärinkäytön tunnistuksesta julkaistussa tutkimuksessa on käsitelty hyvin vähän jäljitystietojen keräämisen vaihtoehtoisia tapoja ja niihin liittyviä ongelmia. Oracle-tietokannanhallintajärjestelmästä saadut kokemukset ovat osittain edellä kuvatulla tavalla sovellettavissa myös muihin tietokannanhallintajärjestelmiin.

Prototyypillä kokeiltiin myös normaalitoiminnan ja -käyttäytymisen profiilien muodostamista aiemmassa tutkimuksessa esitettyjen mallien periaatteita soveltaen. Prototyyppijärjestelmässä tehtiin selkeä ero käyttäjälle tyypillisen toiminnan ja käyttäytymisen mallintamisen välillä. Toimintaprofiili rakennettiin kuvaamaan käyttäjän toimintaa hienojakoisella tasolla Chungin ym. (1999) järjestelmän perusajatusta soveltaen. Käyttöoikeuksien hallintajärjestelmän tapaan rakennettu toimintaprofiili tallettaa ilman redundanssia kaikki käyttäjän opetusvaiheessa suorittamat toiminnot tietokannanhallintajärjestelmän, taulukaavan, kohteen tai attribuutin tasolla. Käyttäytymisprofiili puolestaan tallentaa järjestyksen, jossa käyttäjä toiminnot normaalisti suorittaa. Käyttäytyminen tallennetaan prototyypissä soveltaen Lanen (1998) esittämää ajallisesti järjestettyjen vakiomittaisten jaksojen mallia. Perustana on kuitenkin käytetty toimintaprofiilia, jolloin ylimääräistä tietoa ei tarvitse tallentaa. Käyttäytymisen vertailun perusta on tällöin täsmällisempi kuin Lanen (1998) tai Hofmeyerin ym. (1998) malleissa, joissa tallennetaan jaksojen osiksi vain hyvin vähän jalostettua tietoa.

Käyttäjän toiminnan poikkeamien tunnistamisesta saatiin prototyyppijärjestelmästä hyviä kokemuksia, sillä kaikki järjestelmän kokeiluvaiheessa tehty normaali ja epänormaali tietokantasovellusten käyttö luokiteltiin oikein. Vaikkakin kokeilu tehtiin

hyvin pienellä tapahtumamäärällä, ovat tulokset rohkaisevia, sillä yksinkertaisella periaatteella toimivan järjestelmän voitiin osoittaa toimivan luotettavasti. Prototyyppi-järjestelmän kaltaisen mallin käytännön toimivuuden ja varsinkin sen järjestelmäresurssien käytön taloudellisuuden kokeilu vaatisi kuitenkin koejärjestelyä, joka ei mahdu tämän tutkielman piiriin. Käyttäytymisen poikkeamien tunnistamisen koevaihe antoi vähemmän selviä tuloksia, sillä lyhyestä opetusvaiheesta johtuen väärien positiivisten määrä oli suhteessa korkea. Käyttäytymispoikkeamien tunnistamisen kokeilu vaatisikin laajaa koejärjestelyä mieluusti aidossa ympäristössä.

7 YHTEENVETO

Tämän tutkielman avulla pyrittiin selvittämään millaisia ominaisuuksia relaatiotietokannanhallintajärjestelmän yhteyteen rakennettavalla poikkeamaperusteisella väärinkäytöntunnistamisjärjestelmällä tulisi olla. Erityisesti keskityttiin tällaisen järjestelmän jäljitystietojen keruuseen, normaalitoiminnan ja –käyttäytymisen mallien muodostamiseen sekä varsinaiseen poikkeamien tunnistamiseen edellisten perusteella. Lisäksi käsiteltiin kokonaiskuvan saamiseksi myös väärinkäytöntunnistamisjärjestelmien rakennetta. Tutkimusongelma ositettiin siten, että aluksi selvitettiin millaisia relaatiotietokantojen yhteydessä olennaisia ominaisuuksia jo esitetyissä poikkeamaperusteisissa malleissa on. Toiseksi tutkittiin, mitä erityisesti tietokannanhallintajärjestelmien yhteyteen suunniteluissa poikkeamaperusteisissa malleissa oli esitetty. Lisäksi haluttiin täydentää aiemman julkaistun tutkimuksen epäselviä kohtia rakentamalla konstruktio, jolla pystyttiin selvittämään muun muassa jäljitystietojen keräämisen yksityiskohtia.

Tutkielman alussa käsiteltiin yleisesti aiemmin julkaistua tutkimusta, jonka avulla tutustuttiin väärinkäytöntunnistuksen periaatteisiin, järjestelmien rakenteisiin ja ominaisuuksiin sekä luotiin tutkielmalle käsitepohjaa. Aiemman tutkimuksen tarkastelun avulla pyrittiin löytämään myös ne yleiset poikkeamaperusteisen väärinkäytön tunnistamisen periaatteet, jotka olisivat sovellettavissa myös tietokannanhallintajärjestelmien yhteydessä. Eri tutkimusten vertailusta saatiin käsitteellisesti merkittävä jako käyttäjän toimintaa ja toisaalta käyttäjän käyttäytymistä tunnistaviin malleihin. Vastaavaa jakoa ei ole aiemmissa tutkimuksissa eksplisiittisesti esitetty, vaikka jokaisessa tutkimuksessa suuntautuminen jompaankumpaan lähestymistapaan oli havaittavissa. Tutkielmassa tehty jako on käsitteellisesti merkittävä ja uutta mallia suunniteltaessa olisikin syytä päättää, mitä halutaan mallintaa. Suurin osa aiemmasta yleisestä tutkimuksesta keskittyy järjestelmän normaalitilan tunnistamiseen tai käyttäjän identifioimiseen käyttäytymisen perusteella. Riippumatta siitä, käytetäänkö tunnistamiseen käyttäjän tai järjestelmän toimien temporaalista esitystä, järjestelmästä kerättyjä tilastoja tai käyttäjän biometristä tunnistamista, on kysymys käyttäytymisen tunnistamisesta. Relaatiotietokantojen kontekstissa aiemmin tehty tutkimus keskittyi puolestaan

selvästi käyttäjän toiminnan seuraamiseen. Näistä malleista on löydettävissä yhtäläisyys esimerkiksi käyttöoikeuksien hallintajärjestelmiin.

Julkaistun tutkimuksen perusteella tarkasteltiin myös poikkeamaperusteisen tunnistamisen heikkouksia ja vahvuuksia. Päällimmäisenä vahvuutena poikkeamaperusteisessa väärinkäytöntunnistamisessa on sen periaatteellinen ehdoton kieltävyys, joka takaa sen havaitsevan kaiken sille aiemmin tuntemattoman toiminnan. Sama periaatteellinen ominaisuus on myös heikkous, sillä ehdottomuus aiheuttaa aiemman tutkimuksen mukaan myös paljon vääriä hälytyksiä, jotka vievät luottamuksen järjestelmän toimintaan. Tutkielmassa käsiteltiin vahvuuksien ja heikkouksien tarkastelun perusteella myös toimia, joilla vääriä hälytyksiä voidaan vähentää.

Tutkielma tarkasteli myös yksityiskohtaisemmin tietokannanhallintajärjestelmien yhteyteen ehdotettujen väärinkäytöntunnistusmallien ominaisuuksia. Vaikka aiempaa tutkimusta oli melko vähän, selvä ero yleiseen tutkimukseen oli se, että monimutkaisia rakenteita sisältävien tietokantojen valvontaan tarkoitetut mallit olivat huomattavasti hienojakoisempia. Hienojakoisuus oli osittain viety jopa niin pitkälle, että tietotaso oli otettu mukaan mallintamiseen. Tarkastelun yhteydessä verrattiin yleisen tutkimuksen ja tietokannanhallintajärjestelmiin kohdistuneen tutkimuksen malleja, jonka tuloksena saatiin kerättyä ominaisuuksia, jotka parhaiten soveltuvat väärinkäytön tunnistamiseen tietokannanhallintajärjestelmässä.

Edellä kuvatun käsitteellisen jaon perusteella päädyttiin kahteen tapaan, joista toisella voidaan hienojakoisesti mallintaa käyttäjän toimintaa ja toisella käyttäytymistä toimintojen suorittamisen järjestyksen perusteella. Saatuja tuloksia käytettiin hyväksi toteutettaessa kevyttä prototyyppiä, joka pystyy tunnistamaan väärinkäyttöä yksinkertaisesti ja luotettavasti käyttäjän tallennettua ja uutta toimintaa vertailemalla. Prototyyppiä kokeiltaessa kaikki normaali ja epänormaali toiminta luokiteltiin oikein, mikä on rohkaisevaa vaikka koeympäristö olikin keinotekoinen. Lisäksi järjestelmä havaitsee poikkeamat käyttäjän käyttäytymisessä tallennettua toimintojen suoritusjärjestystä ja uusien jäljitystietojen järjestystä vertailemalla. Prototyyppijärjestelmän rakentamisessa keskityttiin pitkälti jäljitystietojen keräämiseen, sillä Oracle-tietokannanhallintajärjestelmän ominaisuuksia selvitettäessä tämä vaihe osoittautui kaikkein

haasteellisimmaksi. Jäljitystietojen keräämisestä ja siihen liittyvistä ongelmista saatiin tämän ansiosta tutkielman kautta paljon tietoa.

Prototyyppijärjestelmän rakentamisessa ei puututtu väärinkäytön tunnistamisen ongelmiin kohdejärjestelmän resurssikäytön suhteen, eikä väärien positiivisten ongelmaan käyttäytymisen tunnistamisen suhteen. Jatkotutkimusaiheeksi sopisikin sen selvittäminen, miten paljon attribuuttitasolla tapahtuva tunnistaminen varaa tietokannanhallintajärjestelmää. Myös erilaisten rakenneratkaisujen vaikutusta tunnistusjärjestelmän tehokkuuteen ja sen käyttämään tallennustilaan olisi hyvä selvittää. Jatkotutkimuksella olisi myös mielenkiintoista selvittää, kuinka paljon opetustapahtumia suhteessa tietokantasovelluksen kokoon tarvitaan väärien positiivisten tunnistusten minimoimiseksi. Kysymykseen voisi tulla aidon tietokantasovelluksen käyttäjille järjestettävä koe, jossa opetus- ja tunnistusvaiheen tietojen perusteella tutkittaisiin, miten pitkä opetusvaihe vaaditaan väärien positiivisten ongelman minimoimiseksi. Lisäksi järjestelmänvalvojalle tarkoitettujen työtä helpottavien apuvälineiden rakentamista olisi syytä tutkia tarkemmin. Jäljitystietojen keräysasetusten tekemisen tulisi olla yksinkertaista ja nopeaa ja hälytysten tutkimista helpottamalla väärien positiivisten aiheuttama ongelmakaan ei rasittaisi järjestelmänvalvojia liikaa.

LÄHTEET

Allen, J., Christie, A., Fithen, W., McHugh, J., Pickel, J. & Stoner E. 2000. State of the Practise of Intrusion Detection Technologies. Carnegie Mellon Software Engineering Institute, Networked Systems Survivability Program, Technical Report CMU/SEI-99-TR-028, Pittsburgh. Saatavilla www-muodossa <http://www.sei.cmu.edu/pub/documents/99.reports/pdf/99tr028.pdf>.

Anderson, D., Lunt, T. F., Javitz, H., Tamaru, A. & Valdes, A. 1995. Detecting Unusual Program Behavior Using the Statistical Component of the Next-generation Intrusion Detection Expert System (NIDES). SRI International, Computer Science Laboratory, Report SRI-CSL-95-06, Menlo Park. Saatavilla www-muodossa <http://www.sdl.sri.com/projects/nides/reports/5sri.pdf>.

Axelsson, S. 1998. Research in Intrusion-Detection Systems: A Survey. Chalmers University of Technology, Department of Computer Engineering, Technical Report 98-17, Göteborg. Saatavilla www-muodossa <http://www.ce.chalmers.se/staff/sax/survey.ps>.

Axelsson, S. 2000. Intrusion Detection Systems: A Survey and Taxonomy. Chalmers University of Technology, Department of Computer Engineering, Technical Report 99-15, Göteborg. Saatavilla www-muodossa <http://www.ce.chalmers.se/staff/sax/taxonomy.ps>.

Bass, T. 2000. Intrusion Detection Systems and Multisensor Data Fusion. Communication of ACM, 43, 3, 99-105. Saatavilla www-muodossa <http://www.silkroad.com/papers/pdf/acm-p99-bass.pdf>.

Bertino, E., Jajodia, S. & Samarati, P. 1999. A Flexible Authorization Mechanism for Relational Data Management Systems. ACM Transactions on Information Systems 17, 2, 101-140. Saatavilla www-muodossa <http://www.acm.org>.

Carter, D. L. & Katz, A. J. 1996. Computer Crime: An Emerging Challenge for Law Enforcement. FBI Law Enforcement Bulletin December 1996, 1-8. Saatavilla www-muodossa <http://www.fbi.gov/publications/leb/1996/dec961.txt>.

Chung, C., Gertz, M. & Levitt K. 1998. Application-Level Misuse Detection in Relational DBMS. University of California, UC Davis Student Workshop on Computing, Davis. Saatavilla www-muodossa <http://citeseer.nj.nec.com/218943.html>.

Chung, C., Gertz, M. & Levitt K. 1999. DEMIDS: A Misuse Detection System for Database Systems. Teoksessa Third International IFIP TC-11 WG11.5 Working Conference on Integrity and Internal Control in Information Systems, Amsterdam, The Netherlands, November 18-19, 1999, Kluwer Academic Publishers, 159-178. Saatavilla www-muodossa <http://www.db.cs.ucdavis.edu/papers/CGL99b.pdf>.

Denning, D. E. 1987. An Intrusion-Detection Model. IEEE Transactions on Software Engineering 13, 2, 222-232. Saatavilla www-muodossa <http://www.cs.georgetown.edu/~denning/infosec/ids-model.rtf>.

Forrest, S., Hofmeyer, S. A. & Somayaji, A. 1996. Computer Immunology. Communications of ACM 40, 10, 88-96. Saatavilla www-muodossa <http://www.cs.unm.edu/~immsec/publications/cacm96.ps>.

Frank, J. 1994. Artificial Intelligence and Intrusion Detection: Current and Future Directions. Esitetty 17th National Computer Security Conference, Baltimore, October 11-14, 1994. Saatavilla www-muodossa <http:// ic.arc.nasa.gov/people/frank/ncsc.94.ps>.

Hofmeyr, S. A., Forrest, S. & Somayaji A. 1998. Intrusion Detection Using Sequences of System Calls. Journal of Computer Security, 6, 3, 151-180. Saatavilla www-muodossa <http://www.cs.unm.edu/~immsec/publications/ids.ps>.

Ilgun, K., Kemmerer, R. A. & Porras, P. A. 1995. State Transition Analysis: A Rule-Based Intrusion Detection System. IEEE Transactions on Software Engineering, 21, 3,

181-199. Saatavilla www-muodossa <http://www.itoc.usma.edu/ragsdale/refs/stat-TSE.pdf>.

Ingriswang, S. & Liu, P. 2001. AAID: An Application Aware Transaction-Level Database Intrusion Detection System. University of Maryland, Baltimore County, Department of Information Systems, Technical Report 8-01, Baltimore. Saatavilla www-muodossa <http://www.research.umbc.edu/~pliu/pub.html>.

Kumar, S. 1995. Classification and Detection of Computer Intrusions. Purdue University, Department of Computer Science, PhD thesis, West Lafayette. Saatavilla www-muodossa <http://citeseer.nj.nec.com/kumar95classification.html>.

Lane, T. 1998. Machine Learning Techniques for the Computer Security Domain of Anomaly Detection. Purdue University, Department of Electrical and Computer Engineering, Technical Report COAST TR 98-11, West Lafayette. Saatavilla www-muodossa <http://www.cerias.purdue.edu/ssl/techreports-ssl/public/98-11.pdf>.

Lane, T. & Brodley, C. E. 1999. Temporal Sequence Learning and Data Reduction for Anomaly Detection. ACM Transactions on Information and System Security 2, 3, 295-331. Saatavilla www-muodossa <http://www.ai.mit.edu/people/terran/research/pubs/acm_tissec99.pdf>.

Lee, W., Stolfo, S. J. & Mok, K. 1999. A Data Mining Framework for Building Intrusion Detection Models. Teoksessa Proceedings of the 1999 IEEE Symposium on Security and Privacy, Oakland, California, May 9-12, 1999, IEEE Computer Society, 120-132. Saatavilla www-muodossa <http://www.cs.columbia.edu/~sal/hpapers/ieee99.ps.gz>.

Lee, W. & Xiang D. 2001. Information-theoretic Measures for Anomaly Detection. Esitetty 2001 IEEE Symposium on Security and Privacy, The Claremont Resort Oakland, California, May 13-16, 2001. Saatavilla www-muodossa <http://www.csc.ncsu.edu/faculty/lee/papers/entropy-modeling.ps>.

Liu, P. 2001. DAIS: A Real-time Data Attack Isolation System for Commercial Database Applications. Department of Information Systems, UMBC. Esitetty 17th Annual Computer Security Application Conference, New Orleans, December 10-14, 2001. Saatavilla www-muodossa <http://www.acsac.org/2001/papers/44.pdf>.

Lunt, T. F. 1988. Automated Audit Trail Analysis and Intrusion Detection. Esitetty 11th National Computer Security Conference, Baltimore, Maryland, October 17-20, 1988. Saatavilla www-muodossa <http://www.sdl.sri.com/nides/reports/survey88.ps.gz>.

Oracle Corporation 2001a. Oracle 9i Application Developer's Guide – Fundamentals, Release 1 (9.0.1). Saatavilla www-muodossa <http://otn.oracle.com>.

Oracle Corporation 2001b. Oracle 9i Database Administator's Guide, Release 1 (9.0.1). Saatavilla www-muodossa <http://otn.oracle.com>.

Oracle Corporation 2001c. Oracle 9i Database Concepts, Release 1 (9.0.1). Saatavilla www-muodossa <http://otn.oracle.com>.

Oracle Corporation 2001d. Oracle 9i Database Installation Guide, Release 1 (9.0.1.1.1) for Windows. Saatavilla www-muodossa <http://otn.oracle.com>.

Oracle Corporation 2001e. Oracle 9i Database Reference, Release 1 (9.0.1). Saatavilla www-muodossa <http://otn.oracle.com>.

Oracle Corporation 2001f. Oracle 9i Sample Schemas, Release 1 (9.0.1). Saatavilla www-muodossa <http://otn.oracle.com>.

Oracle Corporation 2001g. Oracle 9i SQL Reference, Release 1 (9.0.1). Saatavilla www-muodossa <http://otn.oracle.com>.

Oracle Corporation 2001h. Oracle 9i Supplied PL/SQL Packages and Types Reference, Release 1 (9.0.1). Saatavilla www-muodossa <http://otn.oracle.com>.

Seleznyov, A. & Puuronen S. 1999. Anomaly Intrusion Detection Systems: Handling Temporal Relations between Events. Esitetty 2nd International Workshop on Recent Advances in Intrusion Detection, Lafayette, Indiana, September 7-9, 1999. Saavilla www-muodossa <http://www.raid-symposium.org/raid99/PAPERS/Seleznyov.pdf>.

LIITE 1

Oraclen tietohakemiston taulun AUDIT_ACTIONS sisältö.

Numerotunnus	Toiminta	Numerotunnus	Toiminta
0	UNKNOWN	74	CREATE SNAPSHOT
1	CREATE TABLE	75	ALTER SNAPSHOT
2	INSERT	76	DROP SNAPSHOT
3	SELECT	77	CREATE TYPE
4	CREATE CLUSTER	78	DROP TYPE
5	ALTER CLUSTER	79	ALTER ROLE
6	UPDATE	80	ALTER TYPE
7	DELETE	81	CREATE TYPE BODY
8	DROP CLUSTER	82	ALTER TYPE BODY
9	CREATE INDEX	83	DROP TYPE BODY
10	DROP INDEX	84	DROP LIBRARY
11	ALTER INDEX	85	TRUNCATE TABLE
12	DROP TABLE	86	TRUNCATE CLUSTER
13	CREATE SEQUENCE	91	CREATE FUNCTION
14	ALTER SEQUENCE	92	ALTER FUNCTION
15	ALTER TABLE	93	DROP FUNCTION
16	DROP SEQUENCE	94	CREATE PACKAGE
17	GRANT OBJECT	95	ALTER PACKAGE
18	REVOKE OBJECT	96	DROP PACKAGE
19	CREATE SYNONYM	97	CREATE PACKAGE BODY
20	DROP SYNONYM	98	ALTER PACKAGE BODY
21	CREATE VIEW	99	DROP PACKAGE BODY
22	DROP VIEW	100	LOGON
23	VALIDATE INDEX	101	LOGOFF
24	CREATE PROCEDURE	102	LOGOFF BY CLEANUP
25	ALTER PROCEDURE	103	SESSION REC
26	LOCK	104	SYSTEM AUDIT
27	NO-OP	105	SYSTEM NOAUDIT
28	RENAME	106	AUDIT DEFAULT
29	COMMENT	107	NOAUDIT DEFAULT
30	AUDIT OBJECT	108	SYSTEM GRANT
31	NOAUDIT OBJECT	109	SYSTEM REVOKE
32	CREATE DATABASE LINK	110	CREATE PUBLIC SYNONYM
33	DROP DATABASE LINK	111	DROP PUBLIC SYNONYM
34	CREATE DATABASE	112	CREATE PUBLIC DATABASE LINK
35	ALTER DATABASE	113	DROP PUBLIC DATABASE LINK
36	CREATE ROLLBACK SEG	114	GRANT ROLE
37	ALTER ROLLBACK SEG	115	REVOKE ROLE
38	DROP ROLLBACK SEG	116	EXECUTE PROCEDURE
39	CREATE TABLESPACE	117	USER COMMENT
40	ALTER TABLESPACE	118	ENABLE TRIGGER
41	DROP TABLESPACE	119	DISABLE TRIGGER
42	ALTER SESSION	120	ENABLE ALL TRIGGERS
43	ALTER USER	121	DISABLE ALL TRIGGERS
44	COMMIT	122	NETWORK ERROR
45	ROLLBACK	123	EXECUTE TYPE
46	SAVEPOINT	157	CREATE DIRECTORY
47	PL/SQL EXECUTE	158	DROP DIRECTORY
48	SET TRANSACTION	159	CREATE LIBRARY
49	ALTER SYSTEM	160	CREATE JAVA
50	EXPLAIN	161	ALTER JAVA
51	CREATE USER	162	DROP JAVA
52	CREATE ROLE	163	CREATE OPERATOR
53	DROP USER	164	CREATE INDEXTYPE
54	DROP ROLE	165	DROP INDEXTYPE
55	SET ROLE	167	DROP OPERATOR
56	CREATE SCHEMA	168	ASSOCIATE STATISTICS
57	CREATE CONTROL FILE	169	DISASSOCIATE STATISTICS
59	CREATE TRIGGER	170	CALL METHOD
60	ALTER TRIGGER	171	CREATE SUMMARY
61	DROP TRIGGER	172	ALTER SUMMARY
62	ANALYZE TABLE	173	DROP SUMMARY
63	ANALYZE INDEX	174	CREATE DIMENSION
64	ANALYZE CLUSTER	175	ALTER DIMENSION
65	CREATE PROFILE	176	DROP DIMENSION
66	DROP PROFILE	177	CREATE CONTEXT
67	ALTER PROFILE	178	DROP CONTEXT
68	DROP PROCEDURE	179	ALTER OUTLINE
70	ALTER RESOURCE COST	180	CREATE OUTLINE
71	CREATE SNAPSHOT LOG	181	DROP OUTLINE
72	ALTER SNAPSHOT LOG	182	UPDATE INDEXES
73	DROP SNAPSHOT LOG	183	ALTER OPERATOR

LIITE 2

Proseduuri AUD_SIIRTO.

```
PROCEDURE aud_siirto IS

        V_TIMESTAMP     AUDITSTORE.TIMESTAMP%TYPE;
        V_USERNAME      AUDITSTORE.USERNAME%TYPE;
        V_ACTIONNO      AUDITSTORE.ACTIONNO%TYPE;
        V_SESSIONNO     AUDITSTORE.SESSIONNO%TYPE;
        V_ENTRYNO               AUDITSTORE.ENTRYNO%TYPE;
        V_OBJECTNAME    AUDITSTORE.OBJECTNAME%TYPE;
        V_SCHEMANAME AUDITSTORE.SCHEMANAME%TYPE;

        CURSOR c1 IS
                SELECT TIMESTAMP#, USERID, ACTION#, SESSIONID, ENTRYID, OBJ$NAME,
                OBJ$CREATOR
                FROM SYS.AUD$ LEFT JOIN SYSTEM.AUDITSTORE ON
                SYSTEM.AUDITSTORE.SESSIONNO=SYS.AUD$.SESSIONID AND
                SYSTEM.AUDITSTORE.ENTRYNO=SYS.AUD$.ENTRYID
                WHERE AUDITSTORE.ENTRYNO IS NULL AND RETURNCODE=0 AND
                (ACTION#=1 OR ACTION#=12 OR ACTION#=15 OR ACTION#=21 OR ACTION#=22
                OR ACTION#=24 OR ACTION#=25 OR ACTION#=68 OR ACTION#=116) ORDER BY
                TIMESTAMP#;

BEGIN
        OPEN c1;
        LOOP
                FETCH c1 INTO V_TIMESTAMP, V_USERNAME, V_ACTIONNO, V_SESSIONNO,
                V_ENTRYNO, V_OBJECTNAME, V_SCHEMANAME;
                EXIT WHEN c1%NOTFOUND;
                IF V_ACTIONNO=15 OR V_ACTIONNO=25 OR V_ACTIONNO=116 THEN
                        INSERT INTO AUDITSTORE
                        (TIMESTAMP, USERNAME, ACTIONNO, SESSIONNO, ENTRYNO,
                        OBJECTNAME, SCHEMANAME, COLUMNNAME, AUD_ORDER)
                        VALUES (V_TIMESTAMP, V_USERNAME, V_ACTIONNO, V_SESSIONNO,
                        V_ENTRYNO, V_OBJECTNAME, V_SCHEMANAME,'-',
                        AUD_SEQ.NEXTVAL);
                ELSE
                         INSERT INTO AUDITSTORE
                        (TIMESTAMP, USERNAME, ACTIONNO, SESSIONNO, ENTRYNO,
                        OBJECTNAME, SCHEMANAME, COLUMNNAME, AUD_ORDER)
                        VALUES (V_TIMESTAMP, V_USERNAME, V_ACTIONNO, V_SESSIONNO,
                        V_ENTRYNO, '-', V_SCHEMANAME,'-', AUD_SEQ.NEXTVAL);
                 END IF;
          END LOOP;
          COMMIT;
END;
```

LIITE 3

Proseduuri FGA_SIIRTO.

```
PROCEDURE fga_siirto IS

        V_TIMESTAMP         AUDITSTORE.TIMESTAMP%TYPE;
        V_USERNAME          AUDITSTORE.USERNAME%TYPE;
        V_ACTIONNO          AUDITSTORE.ACTIONNO%TYPE;
        V_SESSIONNO         AUDITSTORE.SESSIONNO%TYPE;
        V_ENTRYNO           AUDITSTORE.ENTRYNO%TYPE;
        V_OBJECTNAME        AUDITSTORE.OBJECTNAME%TYPE;
        V_SCHEMANAME        AUDITSTORE.SCHEMANAME%TYPE;
        V_POLICYNAME        SYS.FGA_LOG$.POLICYNAME%TYPE;
        V_COLUMNNAME        SYS.FGA$.PCOL%TYPE;

        CURSOR c1 IS
        SELECT TIMESTAMP#, DBUID, SESSIONID, SCN, OBJ$NAME, OBJ$SCHEMA,
        POLICYNAME
        FROM SYS.FGA_LOG$ LEFT JOIN SYSTEM.AUDITSTORE ON
        SYSTEM.AUDITSTORE.SESSIONNO=SYS.FGA_LOG$.SESSIONID AND
        SYSTEM.AUDITSTORE.ENTRYNO=SYS.FGA_LOG$.SCN
        WHERE AUDITSTORE.ENTRYNO IS NULL ORDER BY TIMESTAMP#;

BEGIN
        OPEN c1;
        LOOP
                FETCH c1 INTO V_TIMESTAMP, V_USERNAME, V_SESSIONNO, V_ENTRYNO,
                V_OBJECTNAME, V_SCHEMANAME, V_POLICYNAME;
                EXIT WHEN c1%NOTFOUND;
                V_ACTIONNO:=3;
                SELECT PCOL INTO V_COLUMNNAME FROM SYS.FGA$ WHERE
                PNAME=V_POLICYNAME;
                INSERT INTO AUDITSTORE
                (TIMESTAMP, USERNAME, ACTIONNO, SESSIONNO, ENTRYNO, OBJECTNAME,
                SCHEMANAME, COLUMNNAME, AUD_ORDER)
                VALUES (V_TIMESTAMP, V_USERNAME, V_ACTIONNO, V_SESSIONNO,
                V_ENTRYNO, V_OBJECTNAME, V_SCHEMANAME, V_COLUMNNAME,
                AUD_SEQ.NEXTVAL);
         END LOOP;
         COMMIT;
END;
```

LIITE 4

Proseduuri FGA_SS.

```
PROCEDURE fga_ss (pschema VARCHAR2, ptable VARCHAR2, pcolumn VARCHAR2) IS no
NUMBER;

BEGIN
        SELECT COUNT(FGA_ID) INTO no FROM FGA_SETTINGS WHERE
        SCHEMANAME=pschema AND OBJECTNAME=ptable AND COLUMNNAME=pcolumn;
        IF no=0 THEN
                SELECT FGA_SEQ.NEXTVAL INTO no FROM DUAL;
                DBMS_FGA.ADD_POLICY(object_schema =>pschema, object_name =>ptable, pol-
                icy_name => concat('FGA_ID_',no), audit_condition => concat(pcolumn,' LIKE "%"'), au-
                dit_column => pcolumn);
                INSERT INTO FGA_SETTINGS VALUES ('FGA_ID_'||no, pschema, ptable,
                pcolumn);
         ELSE dbms_output.put_line('FGA jo asetettu!'); END IF;
END;
```

LIITE 5

Proseduuri FGA_SR.

```
PROCEDURE fga_sr (pschema VARCHAR2, ptable VARCHAR2, pcolumn VARCHAR2) IS

BEGIN
        dbms_output.put_line(concat(concat(pschema,ptable),pcolumn));
        DBMS_FGA.DROP_POLICY(pschema, ptable, concat(concat(pschema,ptable), pcolumn));
        DELETE FROM FGA_SETTINGS WHERE SCHEMANAME=pschema AND
        OBJECTNAME=ptable AND COLUMNNAME=pcolumn;
END;
```

LIITE 6

Proseduuri FGA_TAB.

```
PROCEDURE fga_tab(set_schema VARCHAR2, set_table VARCHAR2) IS

        V_COLUMN_NAME        SYS.ALL_COL_COMMENTS.COLUMN_NAME%TYPE;
        CURSOR c_col IS
                SELECT COLUMN_NAME FROM SYS.DBA_TAB_COLUMNS WHERE
                OWNER=set_schema AND TABLE_NAME=set_table;

BEGIN
        OPEN c_col;
        LOOP
                FETCH c_col INTO V_COLUMN_NAME;
                EXIT WHEN c_col%NOTFOUND;
                system.fga_ss(set_schema, set_table, v_column_name);
        END LOOP;
        COMMIT;
END;
```

LIITE 7

Proseduuri FGA_REM_TAB.

```
PROCEDURE fga_rem_tab(set_schema VARCHAR2, set_table VARCHAR2) IS

        V_COLUMN_NAME        SYS.ALL_COL_COMMENTS.COLUMN_NAME%TYPE;
        CURSOR c_col IS
                SELECT COLUMN_NAME FROM SYS.DBA_TAB_COLUMNS WHERE
                OWNER=set_schema AND TABLE_NAME=set_table;

BEGIN
         OPEN c_col;
         LOOP
                FETCH c_col INTO V_COLUMN_NAME;
                EXIT WHEN c_col%NOTFOUND;
                system.fga_sr(set_schema, set_table, v_column_name);
         END LOOP;
         COMMIT;
END;
```

LIITE 8

Proseduuri TRIGGER_SS.

```
PROCEDURE trigger_ss (pschema VARCHAR2, ptable VARCHAR2, pcolumn VARCHAR2) IS
komento VARCHAR2(4000); maara NUMBER; no NUMBER;
BEGIN
	SELECT COUNT(TRIG_ID) INTO maara FROM TRIG_SETTINGS WHERE
	SCHEMANAME=pschema AND OBJECTNAME=ptable AND COLUMNNAME='-' AND
	ACTION='INSERT';
	IF maara=0 THEN
		SELECT TRIG_SEQ.NEXTVAL INTO no FROM DUAL;
		komento:='CREATE OR REPLACE TRIGGER TRIG_'||no||' AFTER INSERT ON
		'||pschema||'.'||ptable||' DECLARE audsid NUMBER; entid NUMBER;'||
		' BEGIN SELECT SYS_CONTEXT("userenv","sessionid") INTO audsid FROM DUAL;
		SELECT SYS_CONTEXT("userenv","entryid") INTO entid FROM DUAL;' ||
		' INSERT INTO AUDITSTORE (TIMESTAMP,USERNAME, SESSIONNO, ENTRYNO,
		ACTIONNO, OBJECTNAME, SCHEMANAME, COLUMNNAME, AUD_ORDER) '||'
		VALUES (sysdate, user, audsid, entid, 2, '"||ptable||"', '"||pschema||"',"-",
		AUD_SEQ.NEXTVAL); END;';
		dbms_utility.exec_ddl_statement(komento);
		INSERT INTO TRIG_SETTINGS VALUES ('TRIG_'||no, pschema, ptable, '-','INSERT');
	ELSE dbms_output.put_line('TRIGGER jo asetettu!'); END IF;
	SELECT COUNT(TRIG_ID) INTO maara FROM TRIG_SETTINGS WHERE
	SCHEMANAME=pschema AND OBJECTNAME=ptable AND COLUMNNAME=pcolumn
	AND ACTION='UPDATE';
	IF maara=0 THEN
		SELECT TRIG_SEQ.NEXTVAL INTO no FROM DUAL;
		komento:='CREATE OR REPLACE TRIGGER TRIG_'||no||' AFTER UPDATE OF
		'||pcolumn||' ON '||pschema||'.'||ptable||' DECLARE audsid NUMBER; entid NUMBER;'||
		BEGIN SELECT SYS_CONTEXT("userenv","sessionid") INTO audsid FROM DUAL;
		SELECT SYS_CONTEXT("userenv","entryid") INTO entid FROM DUAL;' ||' INSERT
		INTO AUDITSTORE (TIMESTAMP,USERNAME, SESSIONNO, ENTRYNO,
		ACTIONNO, OBJECTNAME, SCHEMANAME, COLUMNNAME, AUD_ORDER) '||'
		VALUES (sysdate, user, audsid, entid, 6, '"||ptable||"', '"||pschema||"','"||pcolumn||"',
		AUD_SEQ.NEXTVAL); END;';
		dbms_utility.exec_ddl_statement(komento);
		INSERT INTO TRIG_SETTINGS VALUES ('TRIG_'||no, pschema, ptable,
		pcolumn,'UPDATE');
	ELSE dbms_output.put_line('TRIGGER jo asetettu!'); END IF;
		SELECT COUNT(TRIG_ID) INTO maara FROM TRIG_SETTINGS WHERE
		SCHEMANAME=pschema AND OBJECTNAME=ptable AND COLUMNNAME='-'
		AND ACTION='DELETE';
	IF maara=0 THEN
		SELECT TRIG_SEQ.NEXTVAL INTO no FROM DUAL;
		komento:='CREATE OR REPLACE TRIGGER TRIG_'||no||' AFTER DELETE ON
		'||pschema||'.'||ptable||' DECLARE audsid NUMBER; entid NUMBER;'||
		' BEGIN SELECT SYS_CONTEXT("userenv","sessionid") INTO audsid FROM DUAL;
		SELECT SYS_CONTEXT("userenv","entryid") INTO entid FROM DUAL;' ||
		' INSERT INTO AUDITSTORE (TIMESTAMP,USERNAME, SESSIONNO, ENTRYNO,
		ACTIONNO, OBJECTNAME, SCHEMANAME, COLUMNNAME, AUD_ORDER) '||'
		VALUES (sysdate, user, audsid, entid, 7, '"||ptable||"', '"||pschema||"',"-",
		AUD_SEQ.NEXTVAL); END;';
		dbms_utility.exec_ddl_statement(komento);
		INSERT INTO TRIG_SETTINGS VALUES ('TRIG_'||no, pschema, ptable, '-','DELETE');
	ELSE dbms_output.put_line('TRIGGER jo asetettu!'); END IF;
END;
```

LIITE 9

Proseduuri TRIG_TAB.

```
PROCEDURE trig_tab(set_schema VARCHAR2, set_table VARCHAR2) IS

V_COLUMN_NAME                SYS.ALL_COL_COMMENTS.COLUMN_NAME%TYPE;

CURSOR c_col IS
        SELECT COLUMN_NAME FROM SYS.DBA_TAB_COLUMNS WHERE
        OWNER=set_schema AND TABLE_NAME=set_table;

BEGIN
         OPEN c_col;
         LOOP
                FETCH c_col INTO V_COLUMN_NAME;
                EXIT WHEN c_col%NOTFOUND;
                system.trigger_ss(set_schema, set_table, v_column_name);
         END LOOP;
         COMMIT;
END;
```

LIITE 10

Proseduuri ACTION_STORE.

```
PROCEDURE action_store IS

        V_TIMESTAMP         AUDITSTORE.TIMESTAMP%TYPE;
        V_USERNAME          AUDITSTORE.USERNAME%TYPE;
        V_ACTIONNO          AUDITSTORE.ACTIONNO%TYPE;
        V_SESSIONNO         AUDITSTORE.SESSIONNO%TYPE;
        V_ENTRYNO           AUDITSTORE.ENTRYNO%TYPE;
        V_OBJECTNAME        AUDITSTORE.OBJECTNAME%TYPE;
        V_SCHEMANAME        AUDITSTORE.SCHEMANAME%TYPE;
        V_COLUMNNAME        AUDITSTORE.COLUMNNAME%TYPE;

        CURSOR c1 IS
        SELECT DISTINCT S.USERNAME, S.ACTIONNO, S.OBJECTNAME,  S.COLUMNNAME,
        S.SCHEMANAME
        FROM SYSTEM.AUDITSTORE S LEFT JOIN SYSTEM.ACTIONPROFILE P ON
        S.USERNAME = P.USERNAME AND
        S.ACTIONNO = P.ACTIONNO AND
        S.OBJECTNAME = P.OBJECTNAME AND
        S.COLUMNNAME = P.COLUMNNAME AND
        S.SCHEMANAME = P.SCHEMANAME
        WHERE P.USERNAME IS NULL AND S.DONE=0;

BEGIN
        OPEN c1;
        LOOP
                FETCH  c1  INTO  V_USERNAME,  V_ACTIONNO,  V_OBJECTNAME,
                V_COLUMNNAME, V_SCHEMANAME;
                EXIT WHEN c1%NOTFOUND;
                INSERT INTO ACTIONPROFILE
                (PROFILENO,  USERNAME,  ACTIONNO,  OBJECTNAME,  SCHEMANAME,
                COLUMNNAME)
                VALUES  (TEACH_SEQ.NEXTVAL,  V_USERNAME,  V_ACTIONNO,
                V_OBJECTNAME, V_SCHEMANAME, V_COLUMNNAME);
         END LOOP;
         CLOSE c1;
         UPDATE AUDITSTORE SET DONE=1 WHERE DONE=0;
END;
```

LIITE 11

Proseduuri AFFINITY_STORE.

```
PROCEDURE affinity_store IS

V_TIMESTAMP         AUDITSTORE.TIMESTAMP%TYPE;
V_USERNAME          AUDITSTORE.USERNAME%TYPE;
V_ACTIONNO          AUDITSTORE.ACTIONNO%TYPE;
V_SESSIONNO         AUDITSTORE.SESSIONNO%TYPE;
V_ENTRYNO           AUDITSTORE.ENTRYNO%TYPE;
V_OBJECTNAME        AUDITSTORE.OBJECTNAME%TYPE;
V_SCHEMANAME        AUDITSTORE.SCHEMANAME%TYPE;
V_COLUMNNAME        AUDITSTORE.COLUMNNAME%TYPE;
V_PROFILENO         ACTIONPROFILE.PROFILENO%TYPE;
V_TIMES             NUMBER;
V_ENTINEN           NUMBER;

CURSOR c1 IS
      SELECT USERNAME, ACTIONNO, OBJECTNAME, COLUMNNAME, SCHEMANAME,
      PROFILENO FROM SYSTEM.ACTIONPROFILE;



BEGIN
      OPEN c1;
      LOOP
            FETCH c1 INTO V_USERNAME, V_ACTIONNO, V_OBJECTNAME,
            V_COLUMNNAME, V_SCHEMANAME, V_PROFILENO;
            EXIT WHEN c1%NOTFOUND;
                  SELECT COUNT(USERNAME) INTO V_TIMES FROM AUDITSTORE
                  WHERE USERNAME=V_USERNAME AND ACTIONNO=V_ACTIONNO AND
                  SCHEMANAME=V_SCHEMANAME AND OBJECTNAME=V_OBJECTNAME
                  AND COLUMNNAME=V_COLUMNNAME;
                  SELECT TIMES INTO V_ENTINEN FROM ACTIONPROFILE WHERE
                  PROFILENO=V_PROFILENO;
                  V_TIMES:=V_TIMES+V_ENTINEN;
                  UPDATE ACTIONPROFILE SET TIMES=V_TIMES WHERE
            PROFILENO=V_PROFILENO;
            END LOOP;
      CLOSE c1;
      DELETE FROM SYS.FGA_LOG$;
      DELETE FROM SYS.AUD$;
      DELETE FROM AUDITSTORE;
END;
```

LIITE 12

Proseduuri ACTION_STORE2.

```
PROCEDURE action_store2 IS

V_TIMESTAMP          AUDITSTORE.TIMESTAMP%TYPE;
V_USERNAME           AUDITSTORE.USERNAME%TYPE;
V_ACTIONNO           AUDITSTORE.ACTIONNO%TYPE;
V_SESSIONNO          AUDITSTORE.SESSIONNO%TYPE;
V_ENTRYNO            AUDITSTORE.ENTRYNO%TYPE;
V_OBJECTNAME         AUDITSTORE.OBJECTNAME%TYPE;
V_SCHEMANAME         AUDITSTORE.SCHEMANAME%TYPE;
V_COLUMNNAME         AUDITSTORE.COLUMNNAME%TYPE;
V_AUD_ORDER          AUDITSTORE.AUD_ORDER%TYPE;
VB_TEACHNO           ACTIONPROFILE.PROFILENO%TYPE;
VB_TIMESTAMP         AUDITSTORE.TIMESTAMP%TYPE;
VB_USERNAME          AUDITSTORE.USERNAME%TYPE;
VB_ACTIONNO          AUDITSTORE.ACTIONNO%TYPE;
VB_SESSIONNO         AUDITSTORE.SESSIONNO%TYPE;
VB_ENTRYNO           AUDITSTORE.ENTRYNO%TYPE;
VB_OBJECTNAME        AUDITSTORE.OBJECTNAME%TYPE;
VB_SCHEMANAME        AUDITSTORE.SCHEMANAME%TYPE;
VB_COLUMNNAME        AUDITSTORE.COLUMNNAME%TYPE;
V_TRANS_ID           NUMBER;
V_TRANSBACK_ID       NUMBER;
V_FEATURE            VARCHAR(100);
V_FVALUE             VARCHAR(100);
ohita                NUMBER:=0;

    CURSOR c1 IS
        SELECT DISTINCT S.USERNAME, S.ACTIONNO, S.OBJECTNAME,
        S.COLUMNNAME, S.SCHEMANAME, S.AUD_ORDER
        FROM SYSTEM.AUDITSTORE S WHERE DONE=0;

    CURSOR c2 IS SELECT TRANS_ID, FEATURE, FVALUE FROM ACTIONPROFILE2;

BEGIN
    OPEN c1;
    SELECT TRANS_SEQ.NEXTVAL INTO V_TRANS_ID FROM DUAL;
    LOOP
        FETCH c1 INTO V_USERNAME, V_ACTIONNO, V_OBJECTNAME,
        V_COLUMNNAME, V_SCHEMANAME, V_AUD_ORDER;
        EXIT WHEN c1%NOTFOUND;
        open c2;
        LOOP
            FETCH c2 INTO V_TRANSBACK_ID, V_FEATURE, V_FVALUE;
            EXIT WHEN c2%NOTFOUND;
            IF V_FEATURE='USERNAME' THEN
                VB_USERNAME:=V_FVALUE;
            END IF;
            IF V_FEATURE='ACTION' THEN
                VB_ACTIONNO:=V_FVALUE;
            END IF;
            IF V_FEATURE='SCHEMANAME' THEN
                VB_SCHEMANAME:=V_FVALUE;
            END IF;
            IF V_FEATURE='OBJECTNAME' THEN
```

```
                    VB_OBJECTNAME:=V_FVALUE;
                END IF;
                IF V_FEATURE='COLUMNNAME' THEN
                    VB_COLUMNNAME:=V_FVALUE;
                END IF;
                IF V_USERNAME=VB_USERNAME AND V_ACTIONNO = VB_ACTIONNO
                AND V_SCHEMANAME=VB_SCHEMANAME AND
                V_OBJECTNAME=VB_OBJECTNAME AND
                V_COLUMNNAME=VB_COLUMNNAME THEN
                    ohita:=1;
                END IF;
            END LOOP;
            close c2;
            IF ohita=0 THEN
                INSERT INTO ACTIONPROFILE2 (TRANS_ID, FEATURE, FVALUE)
                VALUES (V_TRANS_ID, 'USERNAME', V_USERNAME);
                INSERT INTO ACTIONPROFILE2 (TRANS_ID, FEATURE, FVALUE)
                VALUES (V_TRANS_ID, 'ACTION', V_ACTIONNO);
                INSERT INTO ACTIONPROFILE2 (TRANS_ID, FEATURE, FVALUE)
                VALUES (V_TRANS_ID, 'SCHEMANAME', V_SCHEMANAME);
                INSERT INTO ACTIONPROFILE2 (TRANS_ID, FEATURE, FVALUE)
                VALUES (V_TRANS_ID, 'OBJECTNAME', V_OBJECTNAME);
                INSERT INTO ACTIONPROFILE2 (TRANS_ID, FEATURE, FVALUE)
                VALUES (V_TRANS_ID, 'COLUMNNAME', V_COLUMNNAME);
                UPDATE AUDITSTORE SET DONE=1 WHERE
                AUD_ORDER=V_AUD_ORDER;
            END IF;
        END LOOP;
        CLOSE c1;
END;
```

LIITE 13

Proseduuri REAL_DIST ja sen käyttämät näkymät REF_CON_VIEW, REF_CON_VIEW2 ja REF_CON_VIEW_UNION.

```
PROCEDURE real_dist(dschema VARCHAR2, table1 VARCHAR2, table2 VARCHAR2, distance OUT
NUMBER) IS
        dist NUMBER :=0;
        kierros NUMBER :=1;
        riveja NUMBER:=0;
        ok NUMBER:=0;
        v_owner  VARCHAR2(20);
        v_table1 VARCHAR2(100);
        v_table2 VARCHAR2(100);
        temp1   VARCHAR2(100);
        temp2   VARCHAR2(100);
        v_distno NUMBER;
        v_distance NUMBER;

        CURSOR c1 IS
                SELECT TAB1, TAB2 FROM REF_CON_VIEW_UNION WHERE
                OWNER=dschema;

        CURSOR c2 IS
        SELECT T.DISTNO, V.TAB2
                FROM DIST_TEMP T JOIN REF_CON_VIEW_UNION V ON
                T.TABLE_NAME=V.TAB1 WHERE OWNER=dschema;

        CURSOR c3 IS
        SELECT * FROM DIST_RESULT;


BEGIN
        IF v_table1=v_table2 THEN
                dist:=0;
                ok:=1;
        END IF;
        IF ok=0 THEN
                OPEN c3;
                LOOP
                        FETCH c3 INTO v_table1, v_table2, v_distance, v_owner;
                        EXIT WHEN c3%NOTFOUND;
                        IF table1=v_table1 AND table2=v_table2 AND dschema=v_owner THEN
                                dist:=v_distance;
                                ok:=1;
                                EXIT;
                        END IF;
                END LOOP;
                CLOSE c3;
        END IF;
        IF ok=0 THEN
                OPEN c1;
                kierros:=1;
                LOOP
                        FETCH c1 INTO v_table1, v_table2;
```

```
                EXIT WHEN c1%NOTFOUND;
                IF table1=v_table1 AND table2=v_table2 THEN
                        dist:=1;
                        ok:=1;
                        INSERT INTO DIST_RESULT VALUES (table1, table2, dist, dschema);
                        INSERT INTO DIST_RESULT VALUES (table2, table1, dist, dschema);
                        EXIT;
                END IF;
                IF v_table1=table1 THEN
                        INSERT INTO DIST_TEMP (DISTNO, TABLE_NAME)
                        VALUES (DIST_SEQ.NEXTVAL, v_table2);
                END IF;
        END LOOP;
        CLOSE c1;
        COMMIT;
        IF ok=0 THEN
                <<outer>>
                LOOP
                        kierros:=kierros+1;
                        OPEN c2;
                        SELECT COUNT(T.DISTNO) INTO riveja FROM DIST_TEMP T JOIN
                        REF_CON_VIEW_UNION V ON T.TABLE_NAME=V.TAB1 WHERE
                        OWNER=dschema;
                                LOOP
                                        FETCH c2 INTO v_distno, v_table2;
                                        EXIT WHEN c2%NOTFOUND;
                                        EXIT outer WHEN c2%NOTFOUND AND
                                        c2%ROWCOUNT=0;
                                        IF v_table2=table2 THEN
                                                dist:=kierros;
                                                EXIT OUTER;
                                        END IF;
                                        DELETE FROM DIST_TEMP WHERE DISTNO=v_distno;
                                        INSERT INTO DIST_TEMP (DISTNO, TABLE_NAME)
                                        VALUES (DIST_SEQ.NEXTVAL, v_table2);
                        END LOOP;
                        CLOSE c2;
                        COMMIT;
                END LOOP outer;
                INSERT INTO DIST_RESULT VALUES (table1, table2, dist, dschema);
                INSERT INTO DIST_RESULT VALUES (table2, table1, dist, dschema);
                COMMIT;
        END IF;
    END IF;
    DELETE FROM DIST_TEMP;
    distance:=dist;
    COMMIT;
END real_dist;
```

```
CREATE OR REPLACE VIEW REF_CON_VIEW ( TAB1, TAB2, OWNER ) AS
SELECT A.TABLE_NAME TAB1, B.TABLE_NAME TAB2, A.OWNER OWNER
FROM DBA_CONSTRAINTS A JOIN DBA_CONS_COLUMNS B
ON B.CONSTRAINT_NAME=A.R_CONSTRAINT_NAME WHERE CONSTRAINT_TYPE='R'
```

```
CREATE OR REPLACE VIEW REF_CON_VIEW2 ( TAB1, TAB2, OWNER ) AS
SELECT B.TABLE_NAME TAB1, A.TABLE_NAME TAB2, A.OWNER OWNER
FROM DBA_CONSTRAINTS A JOIN DBA_CONS_COLUMNS B
ON B.CONSTRAINT_NAME=A.R_CONSTRAINT_NAME WHERE CONSTRAINT_TYPE='R'

CREATE OR REPLACE VIEW REF_CON_VIEW_UNION ( TAB1, TAB2, OWNER ) AS
SELECT "TAB1","TAB2","OWNER" FROM REF_CON_VIEW UNION SELECT
"TAB1","TAB2","OWNER" FROM REF_CON_VIEW2 ORDER BY OWNER, TAB1, TAB2
```

LIITE 14

Proseduuri BEHAV_STORE.

```
PROCEDURE behav_store IS

jono                VARCHAR2(100);
maara               NUMBER;
exec_maara          NUMBER := 5;
row_ord             NUMBER := 0;
merkki              NUMBER := 0;
kierroksia          NUMBER := 0;
mismatch            NUMBER := 1;
poistettava         NUMBER;
entinen             NUMBER := 0;
V_AUD_ORDER         AUDITSTORE.AUD_ORDER%TYPE;
V_TIMESTAMP         AUDITSTORE.TIMESTAMP%TYPE;
V_USERNAME          AUDITSTORE.USERNAME%TYPE;
V_ACTIONNO          AUDITSTORE.ACTIONNO%TYPE;
V_SESSIONNO         AUDITSTORE.SESSIONNO%TYPE;
V_ENTRYNO           AUDITSTORE.ENTRYNO%TYPE;
V_OBJECTNAME        AUDITSTORE.OBJECTNAME%TYPE;
V_SCHEMANAME        AUDITSTORE.SCHEMANAME%TYPE;
V_COLUMNNAME        AUDITSTORE.COLUMNNAME%TYPE;
VB_TEACHNO          ACTIONPROFILE.PROFILENO%TYPE;
VB_TIMESTAMP        AUDITSTORE.TIMESTAMP%TYPE;
VB_USERNAME         AUDITSTORE.USERNAME%TYPE;
VB_ACTIONNO         AUDITSTORE.ACTIONNO%TYPE;
VB_SESSIONNO        AUDITSTORE.SESSIONNO%TYPE;
VB_ENTRYNO          AUDITSTORE.ENTRYNO%TYPE;
VB_OBJECTNAME       AUDITSTORE.OBJECTNAME%TYPE;
VB_SCHEMANAME       AUDITSTORE.SCHEMANAME%TYPE;
VB_COLUMNNAME       AUDITSTORE.COLUMNNAME%TYPE;
V_USERS             VARCHAR(40);

CURSOR users IS
    SELECT DISTINCT USERNAME FROM AUDITSTORE WHERE DONEB=0;

CURSOR c1 IS
    SELECT TIMESTAMP, USERNAME, ACTIONNO, COLUMNNAME, OBJECTNAME,
    SCHEMANAME, AUD_ORDER FROM AUDITSTORE WHERE DONEB=0 AND
    USERNAME=v_users ORDER BY AUD_ORDER;

CURSOR c2 IS
    SELECT PROFILENO, USERNAME, ACTIONNO, COLUMNNAME, OBJECTNAME,
    SCHEMANAME FROM ACTIONPROFILE WHERE USERNAME=v_users;

BEGIN
    OPEN users;
    LOOP
        EXIT WHEN users%NOTFOUND;
        FETCH users INTO v_users;
        SELECT COUNT(USERNAME) INTO maara FROM AUDITSTORE WHERE
        DONEB=0 AND USERNAME=v_users;
        IF maara >= exec_maara THEN
            kierroksia := maara-exec_maara+1;
```

```
                FOR i IN 1..kierroksia LOOP
                        jono:='';
                        OPEN c1;
                        FOR j IN 1..exec_maara LOOP
                                FETCH c1 INTO V_TIMESTAMP, V_USERNAME,
                                V_ACTIONNO, V_COLUMNNAME, V_OBJECTNAME,
                                V_SCHEMANAME, V_AUD_ORDER;
                                EXIT WHEN c1%NOTFOUND;
                                OPEN c2;
                                        LOOP
                                                FETCH c2 INTO VB_TEACHNO, VB_USERNAME,
                                                VB_ACTIONNO, VB_COLUMNNAME,
                                                VB_OBJECTNAME, VB_SCHEMANAME;
                                                EXIT WHEN c2%NOTFOUND;
                                                IF V_USERNAME=VB_USERNAME AND
                                                V_ACTIONNO=VB_ACTIONNO AND
                                                V_COLUMNNAME=VB_COLUMNNAME AND
                                                V_OBJECTNAME=VB_OBJECTNAME AND
                                                V_SCHEMANAME=VB_SCHEMANAME THEN
                                                        jono:=concat(jono, VB_TEACHNO);
                                                        jono:=concat(jono, ' ');
                                                END IF;
                                                EXIT WHEN V_USERNAME=VB_USERNAME
                                                AND V_ACTIONNO=VB_ACTIONNO AND
                                                V_COLUMNNAME=VB_COLUMNNAME AND
                                                V_OBJECTNAME=VB_OBJECTNAME AND
                                                V_SCHEMANAME=VB_SCHEMANAME;
                                        END LOOP;
                                        IF j=1 THEN
                                                UPDATE AUDITSTORE SET DONEB=1 WHERE
                                                AUD_ORDER=V_AUD_ORDER;
                                        END IF;
                                        CLOSE c2;
                                END LOOP;
                                CLOSE c1;
                                SELECT COUNT(USERNAME) INTO maara FROM
                                BEHAVPROFILE WHERE BEHAVIOUR=jono;
                                IF maara=0 THEN
                                        INSERT INTO BEHAVPROFILE (BEHAVNO,
                                        USERNAME, BEHAVIOUR) VALUES
                                        (BEHAV_SEQ.NEXTVAL, v_users, jono);
                                ELSE
                                        SELECT TIMES INTO entinen FROM BEHAVPROFILE
                                        WHERE BEHAVIOUR=jono;
                                        entinen:=entinen+1;
                                        UPDATE BEHAVPROFILE SET TIMES=entinen WHERE
                                        BEHAVIOUR=jono;
                                END IF;
                        END LOOP;
                END IF;
        END LOOP;
        close users;
END;
```

LIITE 15

Proseduuri DETECT_ACTION.

```
PROCEDURE detect_action IS

V_TIMESTAMP         AUDITSTORE.TIMESTAMP%TYPE;
V_USERNAME          AUDITSTORE.USERNAME%TYPE;
V_ACTIONNO          AUDITSTORE.ACTIONNO%TYPE;
V_SESSIONNO         AUDITSTORE.SESSIONNO%TYPE;
V_ENTRYNO           AUDITSTORE.ENTRYNO%TYPE;
V_OBJECTNAME        AUDITSTORE.OBJECTNAME%TYPE;
V_SCHEMANAME        AUDITSTORE.SCHEMANAME%TYPE;
V_COLUMNNAME        AUDITSTORE.COLUMNNAME%TYPE;

CURSOR c1 IS
      SELECT S.TIMESTAMP, S.USERNAME, S.ACTIONNO, S.OBJECTNAME,
      S.COLUMNNAME, S.SCHEMANAME, S.SESSIONNO
      FROM SYSTEM.AUDITSTORE S LEFT JOIN SYSTEM.ACTIONPROFILE P ON
      S.USERNAME = P.USERNAME AND
      S.ACTIONNO = P.ACTIONNO AND
      S.OBJECTNAME = P.OBJECTNAME AND
      S.COLUMNNAME = P.COLUMNNAME AND
      S.SCHEMANAME = P.SCHEMANAME
      WHERE P.USERNAME IS NULL AND S.DONE=0;

BEGIN
      OPEN c1;
      LOOP
            FETCH c1 INTO V_TIMESTAMP, V_USERNAME, V_ACTIONNO,
            V_OBJECTNAME, V_COLUMNNAME, V_SCHEMANAME, V_SESSIONNO;
            EXIT WHEN c1%NOTFOUND;
            INSERT INTO ALARMS
            (TIMESTAMP, USERNAME, ACTIONNO, OBJECTNAME, SCHEMANAME,
            COLUMNNAME, SESSIONNO)
            VALUES (V_TIMESTAMP, V_USERNAME, V_ACTIONNO, V_OBJECTNAME,
            V_SCHEMANAME, V_COLUMNNAME, V_SESSIONNO);
       END LOOP;
       CLOSE c1;
       UPDATE AUDITSTORE SET DONE=1 WHERE DONE=0;
END;
```

LIITE 16

Proseduuri DETECT_BEHAV.

```
PROCEDURE detect_behav IS

jono                VARCHAR2(100);
maara               NUMBER;
exec_maara          NUMBER := 5;
row_ord             NUMBER := 0;
merkki              NUMBER := 0;
kierroksia          NUMBER := 0;
mismatch            NUMBER := 1;
poistettava         NUMBER;
det_user            VARCHAR(40);
V_AUD_ORDER         AUDITSTORE.AUD_ORDER%TYPE;
V_TIMESTAMP         AUDITSTORE.TIMESTAMP%TYPE;
V_USERNAME          AUDITSTORE.USERNAME%TYPE;
V_ACTIONNO          AUDITSTORE.ACTIONNO%TYPE;
V_SESSIONNO         AUDITSTORE.SESSIONNO%TYPE;
V_ENTRYNO           AUDITSTORE.ENTRYNO%TYPE;
V_OBJECTNAME        AUDITSTORE.OBJECTNAME%TYPE;
V_SCHEMANAME        AUDITSTORE.SCHEMANAME%TYPE;
V_COLUMNNAME        AUDITSTORE.COLUMNNAME%TYPE;
VB_TEACHNO          ACTIONPROFILE.PROFILENO%TYPE;
VB_TIMESTAMP        AUDITSTORE.TIMESTAMP%TYPE;
VB_USERNAME         AUDITSTORE.USERNAME%TYPE;
VB_ACTIONNO         AUDITSTORE.ACTIONNO%TYPE;
VB_SESSIONNO        AUDITSTORE.SESSIONNO%TYPE;
VB_ENTRYNO          AUDITSTORE.ENTRYNO%TYPE;
VB_OBJECTNAME       AUDITSTORE.OBJECTNAME%TYPE;
VB_SCHEMANAME       AUDITSTORE.SCHEMANAME%TYPE;
VB_COLUMNNAME       AUDITSTORE.COLUMNNAME%TYPE;

CURSOR c1 IS
    SELECT TIMESTAMP, USERNAME, ACTIONNO, COLUMNNAME, OBJECTNAME,
    SCHEMANAME, AUD_ORDER FROM AUDITSTORE WHERE DONEB=0  AND
    USERNAME=det_user ORDER BY AUD_ORDER;

CURSOR c2 IS
    SELECT PROFILENO, USERNAME, ACTIONNO, COLUMNNAME, OBJECTNAME,
    SCHEMANAME FROM ACTIONPROFILE WHERE USERNAME=det_user;

CURSOR users IS
    SELECT DISTINCT USERNAME FROM AUDITSTORE WHERE DONEB=0;

BEGIN
    OPEN users;
    LOOP
        EXIT WHEN users%NOTFOUND;
        FETCH users INTO det_user;
        SELECT COUNT(USERNAME) INTO maara FROM AUDITSTORE WHERE
        DONEB=0 AND USERNAME=det_user;
        IF maara >= exec_maara THEN
            kierroksia := maara-exec_maara+1;
            FOR i IN 1..kierroksia LOOP
```

```
                    jono:='';
                    open c1;
                    FOR j IN 1..exec_maara LOOP
                        FETCH c1 INTO V_TIMESTAMP, V_USERNAME,
                        V_ACTIONNO, V_COLUMNNAME, V_OBJECTNAME,
                        V_SCHEMANAME, V_AUD_ORDER;
                        EXIT WHEN c1%NOTFOUND;
                        OPEN c2;
                        LOOP
                            FETCH c2 INTO VB_TEACHNO, VB_USERNAME,
                            VB_ACTIONNO, VB_COLUMNNAME,
                            VB_OBJECTNAME, VB_SCHEMANAME;
                            EXIT WHEN c2%NOTFOUND;
                            IF V_USERNAME=VB_USERNAME AND
                            V_ACTIONNO=VB_ACTIONNO AND
                            V_COLUMNNAME=VB_COLUMNNAME AND
                            V_OBJECTNAME=VB_OBJECTNAME AND
                            V_SCHEMANAME=VB_SCHEMANAME THEN
                                jono:=concat(jono, VB_TEACHNO);
                                jono:=concat(jono, ' ');
                            END IF;
                            EXIT WHEN V_USERNAME=VB_USERNAME AND
                            V_ACTIONNO=VB_ACTIONNO AND
                            V_COLUMNNAME=VB_COLUMNNAME AND
                            V_OBJECTNAME=VB_OBJECTNAME AND
                            V_SCHEMANAME=VB_SCHEMANAME;
                        END LOOP;
                        IF j=1 THEN UPDATE AUDITSTORE SET DONEB=1 WHERE
                        AUD_ORDER=V_AUD_ORDER; END IF;
                        CLOSE c2;
                    END LOOP;
                    CLOSE c1;
                    SELECT COUNT(USERNAME) INTO maara FROM BEHAVPROFILE
                    WHERE BEHAVIOUR=jono;
                    IF maara=0 THEN
                        INSERT INTO ALARMS (USERNAME, TIMESTAMP,
                        BEHAVIOUR) VALUES (det_user, sysdate, jono);
                    END IF;
                END IF;
            END LOOP;
        END LOOP;
        close users;
END;
```

LIITE 17

Hälytysten tarkastelunäkymien ACTION_ALARMS, ACTION_ALARMS_DIST, ACTION_ALARMS_DIST2 ja BEHAV_ALARMS määrittelylauseet.

```
CREATE OR REPLACE VIEW ACTION_ALARMS ( TIMESTAMP, USERNAME, SCHEMANAME, OBJECTNAME, COLUMNNAME, ACTIONNO, SESSIONNO ) AS
SELECT DISTINCT TIMESTAMP, "USERNAME", "SCHEMANAME", "OBJECTNAME", "COLUMNNAME", "ACTIONNO", "SESSIONNO" FROM ALARMS WHERE BEHAVIOUR IS NULL ORDER BY TIMESTAMP, USERNAME, OBJECTNAME DESC

CREATE OR REPLACE VIEW ACTION_ALARMS_DIST ( LAST_TIMESTAMP, USERNAME, SCHEMANAME, OBJECTNAME, COLUMNNAME, ACTIONNO, SESSIONNO, TIMES ) AS
SELECT MAX(TIMESTAMP), "USERNAME", "SCHEMANAME", "OBJECTNAME", "COLUMNNAME", "ACTIONNO", SESSIONNO, COUNT(USERNAME) FROM ALARMS WHERE BEHAVIOUR IS NULL GROUP BY USERNAME, SCHEMANAME, OBJECTNAME, COLUMNNAME, ACTIONNO, SESSIONNO ORDER BY USERNAME, OBJECTNAME DESC

CREATE OR REPLACE VIEW ACTION_ALARMS_DIST2 ( LAST_TIMESTAMP, USERNAME, SCHEMANAME, OBJECTNAME, COLUMNNAME, ACTIONNO, LAST_SESSIONNO, TIMES ) AS
SELECT MAX(TIMESTAMP), "USERNAME", "SCHEMANAME", "OBJECTNAME", "COLUMNNAME", "ACTIONNO", MAX(SESSIONNO), COUNT(USERNAME) FROM ALARMS WHERE BEHAVIOUR IS NULL GROUP BY USERNAME, SCHEMANAME, OBJECTNAME, COLUMNNAME, ACTIONNO ORDER BY USERNAME, OBJECTNAME DESC

CREATE OR REPLACE VIEW BEHAV_ALARMS ( TIMESTAMP, USERNAME, SESSIONNO, BEHAVIOUR ) AS SELECT "TIMESTAMP", "USERNAME", "SESSIONNO", "BEHAVIOUR" FROM ALARMS WHERE BEHAVIOUR IS NOT NULL ORDER BY TIMESTAMP, USERNAME, OBJECTNAME DESC
```